GeoCOM
Reference Manual

1 Introduction 1-1
1.1 TPS1000 System Software 1-1
1.2 Principles of GeoCOM Operation 1-3
2 General Concepts of Using GeoCOM 2-1
2.1 General Concept of Operation 2-1
2.2 ASCII Protocol 2-2
2.3 Function Call Protocol – C/C++ 2-4
2.4 Function Call Protocol – VBA 2-5
3 Fundamentals of Programming GeoCOM 3-1
3.1 ASCII Protocol Programming 3-1
3.2 C/C++ - Programming 3-5
3.3 VBA – Programming 3-8
3.4 Units of Values 3-13
3.5 TPS1000 Instrument Modes of Operation 3-14
3.6 Common Communication Errors 3-16
4 Remarks on the Description 4-1
4.1 Structure of Descriptions 4-1
5 Communication Settings 5-1
5.1 Constants and Types 5-1
5.2 General GeoCOM Functions 5-2
5.3 Client Specific GeoCOM Functions 5-5
6 Automation - AUT 6-1
6.1 Usage 6-1
6.2 Constants and Types 6-1
6.3 Functions 6-3
7 Basic Applications - BAP 7-1
7.1 Constants and Types 7-1
7.2 Functions 7-1
8 Basic Man Machine Interface - BMM 8-1
8.1 Constants and Types 8-1
8.2 Functions 8-1
9 Communications - COM 9-1
9.1 Constants and Types 9-1
9.2 Functions 9-1
10 Central Services - CSV 10-1
10.1 Usage 10-1
10.2 Constants and Types 10-1
10.3 Functions 10-3

11 Controller Task – CTL 11-1
11.1 Functions 11-1
12 Electronic Distance Measurement - EDM 12-1
12.1 Usage 12-1
12.2 Constants and Types 12-1
12.3 Functions 12-2
13 Motorisation - MOT 13-1
13.1 Usage 13-1
13.2 Constants and Types 13-1
13.3 Functions 13-2
14 Supervisor - SUP 14-1
14.1 Usage 14-1
14.2 Constants and Types 14-1
14.3 Functions 14-2
15 Theodolite Measurement and Calculation - TMC 15-1
15.1 Usage 15-2
15.2 Constants and Types 15-3
15.3 Measurement Functions 15-7
15.4 Measurement Control Functions 15-22
15.5 Data Setup Functions 15-26
15.6 Information Functions 15-39
15.7 Configuration Functions 15-41
16 WI - Registration – WIR 16-1
16.1 Constants 16-1
16.2 Functions 16-1

A Return Codes A-1
A-1 General Return Codes A-1
A-2 ANG Subsystem A-2
A-3 ATA Subsystem A-3
A-4 EDM Subsystem A-3
A-5 TMC Subsystem A-4
A-6 MOT Subsystem A-4
A-7 MOT Subsystem A-5
A-8 WIR Subsystem A-6
A-9 AUT Subsystem A-7
A-10 BAP Subsystem A-7

B Hardware interface B-1
B-1 Serial Interface B-1
B-2 Debugging Utility B-2
C Provided Samples C-1
C-1 Settings for Terminal Emulator C-1
C-2 Program Frames C-1
D List of RPC's D-1
D-1 Alpha Order D-1
D-2 Numeric Order D-3

1 INTRODUCTION

TPS1000 series Theodolites are modern geodetic measurement instruments. Most of the main tasks can be fulfilled with these instruments implicitly by their integrated applications. Now, to fulfil a broader spectrum of tasks and applications an interface to the TPS1000 series sensor functions has been defined and will be published with this document.

With this interface it will be possible to write client applications based on MS-Windows and/or for any other platform which supports ASCII based communications.

1.1 TPS1000 SYSTEM SOFTWARE

The TPS1000 system software organises and controls the interplay of several sensor elements. Furthermore, it builds up a frame for applications, which can be executed on the TPS1000 Theodolite.

This document concentrates on the main interface to the sensor elements of the TPS1000 Theodolite. This main interface can be used to implement solutions for special customer problems if the already existing solution does not provide the needed functionality or just to enhance it.

1.1.1 Organisation of Subsystems

The TPS1000 system software is built around the sensor elements, which are parts and/or optional add-ons of the TPS1000 Theodolite instrument. It provides a set of functions to access sensors and calculated values. These functions are organised as subsystems. We will keep this segmentation in this document.

These functions can be grouped in the following sections:

AUT Automatisation; a module which provides functions like the control of the Automatic Target Recognition, Change Face function or Positioning functions.

BAP Basic Applications; some functions which can easily be used to get measuring data.

BMM Basic Man Machine; functions which controls some basic input/output functionality, e.g. set beep alarm, etc.

COMF Communication; a module which handles the basic communication parameters. Most of these functions relate to both client and server side.

COM Communication; functions to access some aspects of TPS1000 control which are close to communication. These functions relate either to the client side or to the server side.

CSV Central Services; this module provides functions to get or set central/basic information about the TPS1000 instrument.

CTL Control task; this module contains functions of the system control task.

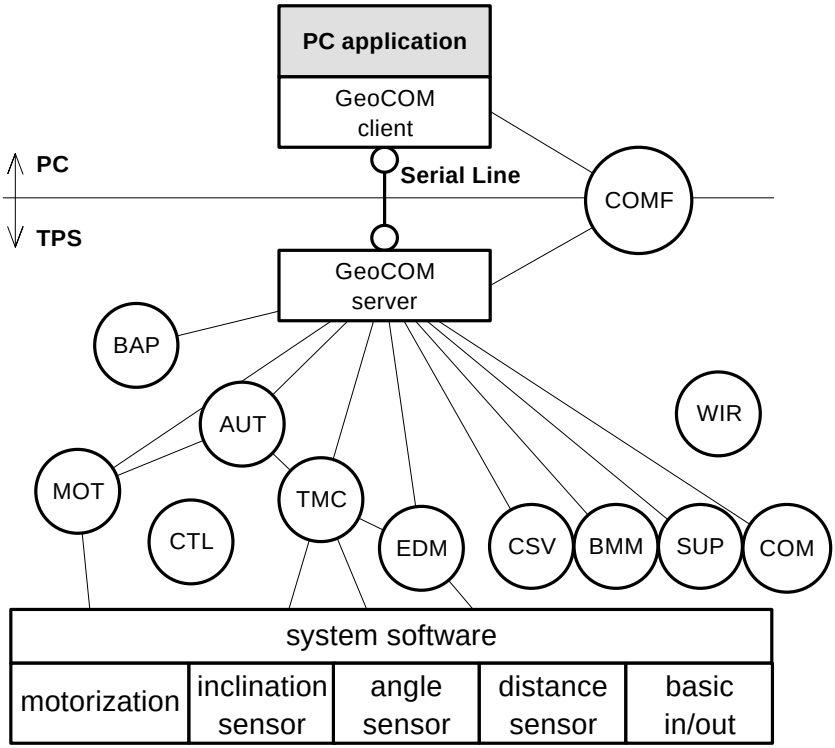
EDM Electronic Distance Meter; the module which measures distances.

MOT Motorization; the part which can be used to control the movement and the speed of movements of the instrument.

SUP Supervisor; functions to control some of the general values of the TPS1000 instrument, e.g. set the lower limit temperature.

TMC Theodolite Measurement and Calculation; the core module for getting measurement data.

WIR WI Registration; this module contains function for GSI recording.



Picture (1) - Overview Client/Server Application

1.2 PRINCIPLES OF GEOCOM OPERATION

Communication takes place between two participants – a client and a server. The medium of communication is a serial communication line. Refer to Appendix B for further information about settings and needed hardware.

The idea of GeoCOM is based on SUN Microsystems' Remote Procedure Call (RPC) protocol.

On the low level of implementation, each procedure, which is executable on the remote instrument, is assigned a remote procedure call identification number. This number is used internally to associate a specific request, including the implicit

parameters, to a procedure on the remote device. On this level, GeoCOM provides an ASCII interface, which can be used to implement applications on platforms, which do not support MS-Windows.

On the high level, GeoCOM provides normal function call interfaces for C/C++ and MS-VBA to these remote functions. These interfaces enable a programmer to implement an application as if it would be executed directly on the TPS1000 instrument.

Note: Further on we will refer to a remotely executable system function as a *RPC*.

The TPS1000 instrument system software uses a multitasking operating system. Nevertheless, only one request can be executed at once. This means in respect of calling RPC's GeoCOM works synchronously only.

On the low level interface the server buffers subsequent requests if current request(s) has not been finished so far. If the queue is full then subsequent requests will be lost.

Instead on the high level interface a function call will not return until it has been completely finished.

2 GENERAL CONCEPTS OF USING GEOCOM

Here we will describe several aspects of using GeoCOM. One of them is how to execute a function at a TPS1000 instrument.

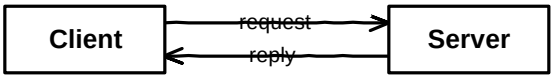
The current implementation of GeoCOM supports two (three) kinds of usage. We can distinguish between a rather rudimentary ASCII protocol and a high level function call interface.

The former - ASCII protocol - is made up of requests and replies. Using GeoCOM in this way means that an application assembles a request, sends it over the serial line to the listening TPS1000 instrument, wait for the answer and decode the received reply.

The latter uses normal function calls either in C/C++ or in VBA. For explanation purposes we will split it into two categories because the two supported programming environments differ in relation to their type systems. Using GeoCOM in this way means calling a function. Any necessary communication will be handled by GeoCOM implicitly.

2.1 GENERAL CONCEPT OF OPERATION

Fundamentally, GeoCOM is implemented as a point to point communication system. The two communication participants are known as the client (external device) and the server (TPS1000 instrument). One communication unit consists of a request and a corresponding reply. Hence, one communication takes place when the client sends a request to the server and the server sends a reply back to the client.



Picture 2-1: Basic communication

GeoCOM is implemented as synchronous communication. A request/reply pair may no be interrupted by another request/reply. Instead, a communication unit must be completed successfully before a new communication unit may be initiated.

Although the ASCII protocol allows sending the next request before the corresponding reply has been received, it is not recommended to do that. Of course, subsequent request will be buffered when the previous request has not been finished so far. But if the buffer content reaches its limit in size then data may be lost.

Note: In the current implementation, only one communication channel per session will be supported. Hence, only one instrument can be connected at a time. Nevertheless, the nature of the ASCII protocol makes it possible to connect and communicate to more than one instrument at a time.

2.2 ASCII PROTOCOL

In sequence we will define the syntax first and then give some information about how to use the ASCII protocol to call a function on the TPS1000 instrument.

The ASCII protocol is a line protocol, hence it uses a line terminator to distinguish between different requests (replies). One request must be terminated by one terminator.

2.2.1 ASCII Protocol Syntax

Syntax of an ASCII request:

```
[<LF>]%R1Q,<RPC>[,<TrId>]:[<P0>][,<P1>,...]<Term>
```

Optional items are in brackets []. The angled-brackets <> surround names or descriptions. These names have variable values depending on their types and meanings. The angled-brackets themselves are not part of the transferred text. Characters not surrounded by brackets are literal text and are part of the GeoCOM protocol.

<LF>	An initial line feed clears the receiver buffer.
%R1Q	GeoCOM request type 1.
<RPC>	Remote Procedure Call identification number in between 0 to 65535.

<TrId>	Optional transaction ID: normally incremented from 1 to 7. Same value in reply.
:	Separator between protocol header and following parameters.
<P0>,<P1>,...	Parameter 0, Parameter 1, ...
<Term>	Terminator string (default CR/LF, use COM_SetTerminator to change the terminator). As a common shortcut '^m' will be used in examples.

Example:

The following example uses the RPC CSV_SetUserInstrumentName to set a new user instrument name ('^m' denotes the terminator):

```
%R1Q,5005:"New Instrument Name"^m
```

Note: Additional characters at the beginning of a request, between parameters or at the end are not allowed. They might lead to errors during interpretation.

Syntax of an ASCII reply:

```
%R1P,<GRC>[,<TrId>]:<RC>[,<P0>,<P1>, ...]<Term>
```

Optional items are in brackets []. The angled-brackets <> surround names or descriptions. These names have variable values as described in the types they have. The angled-brackets themselves are not a part of the communication text. Characters not surrounded by angled-brackets are literal text and are part of the GeoCOM protocol.

%R1P	GeoCOM reply type 1.
<GRC>	GeoCOM return code. This value denotes the success of the communication. 0 = RC_OK means the communication was successful (see table 'GeoCOM return codes' in the appendix for further information).
<TrId>	Transaction ID – identical to that of the request. If the request had no Transaction ID then it will be 0.
:	Separator between protocol header and following parameters.

<RC>	Return code from the called RPC and denotes the successful completion if it is set to 0 (see table 'RPC return codes' in the appendix for further information).
<P0>,<P1>,...	Parameter 0, Parameter 1, … These parameters will be valid only if <GRC> is equal to 0 (RC_OK).
<Term>	Terminator string (default CR/LF, use COM_SetTerminator to change the terminator).

Example:

The following example shows the reply to the RPC 5008 – CSV_GetDateTime.

```
%R1P,0,0:0,1996,'07','19','10','13','2f'^m
     ¦ ¦ ¦ ----------------------------
     ¦ ¦ ¦       ¦      The values for month, day, hour,
     ¦ ¦ ¦       +---   minute and second are replied in the byte-
     ¦ ¦ ¦              format (see table communication parameter
     ¦ ¦ ¦              for further information)
     ¦ ¦ +------  Return code from the RPC: 0 means  no error
     ¦ ¦           (see RPC return codes for further  information)
     ¦ +-----  The Transaction ID of the request. If there was  no ID
     ¦          the value returned is 0.
     +----  Return code from GeoCOM: 0 means no error  (see
            GeoCOM return codes for further information)
```

2.3 FUNCTION CALL PROTOCOL – C/C++

The implementation of GeoCOM for C/C++ conforms to normal function calls. GeoCOM itself handles all necessary communication. No intervention of the programmer in respect to the communication is necessary with one exception. If the GeoCOM reports a communication error the programmer has to make sure that either the problem will be solved - by calling GeoCOM support functions - or no further RPC's will be called – by terminating the running task.

Nevertheless, the programmer has to initialise GeoCOM and set up the port's settings to make sure that communication can take place. Moreover the user has to make sure that the TPS1000 instrument is well connected.

Example:

An example code fragment for using `TMC_GetSimpleMea` could be the following. We do not take care of the necessary initialisation and set up of GeoCOM here. Please refer to chapter 3.2.3 Basic GeoCOM Application Frame for C/C++ for this information.

```
{
  RC_TYPE       RetCode;
  TMC_HZ_V_ANG  Angles;
  double        dSlopeDist;

  RetCode = TMC_GetSimpleMea( 1000, Angles,
                                    dSlopeDist,
                                    TMC_AUTO_INC );
  if (RetCode == RC_OK)
  {
    // do something – use values
  }
  else
  {
    // handle error
  }
}
```

2.4 FUNCTION CALL PROTOCOL – VBA

Here almost all is valid for VBA as for C/C++. Please refer to Chapter 2.3. The only difference between VBA and C/C++ is that VBA has a different type system. Hence, the defined data types differ slightly in their definition. Furthermore, because of implementation reasons the RPC names must have an additional prefix, which is "VB_" for the current implementation of GeoCOM.

Example:

We take the same example as in Chapter 2.3.

```
Dim RetCode     As Integer
Dim Angles      As TMC_HZ_V_ANG
Dim dSlopeDist  As Double
RetCode = VB_TMC_GetSimpleMea( 1000, Angles,
                               dSlopeDist,
                               TMC_AUTO_INC )
If RetCode = RC_OK Then
  ' do something – use values
Else
  ' handle error
```

3 FUNDAMENTALS OF PROGRAMMING GEOCOM

We will describe how programs can be written using the different protocols. Certainly, the type system, where the main differences lie between the protocols, will be described in more detail.

3.1 ASCII PROTOCOL PROGRAMMING

Implementing an application, which uses the ASCII protocol, is based on simple data transfers using a serial line. The programmer is responsible to set up the serial line parameters of the client such that they correspond to the settings of the TPS1000 instrument. Then Remote calls are done by just sending the valid encoded requests and receiving and decoding the replies of them.

For debugging purposes, it might be helpful to use a so-called Y-cable, which enables you to observe the communication on the serial line using either a terminal or a terminal emulator. For further details see Appendix **B-2 Debugging Utility**.

Note: If the settings of the active COM port will be set by any software part and if the server is online, then it is strongly recommended to use a leading <LF> to clear the receiver buffer at the server side. This will reduce unnecessary error messages of the next RPC.

3.1.1 Data Types in ASCII Protocol

Each parameter of a RPC has its own associated data type with it. There are varieties of different data types, which have been defined for the set of published functions. The ASCII protocol supports simple data types only. All data types, which are different from the base, types in name and aggregated data types are converted and reduced to there base types. Conversion means to serialise the aggregated data into a comma-separated list of its elements. Therefore, the programmer has the responsibility to interpret the values depending on the associated data type.

The supported base types and their value range are defined below:

Format Type	Valid range	Len	Valid input representations	Typical output representations
`boolean`	0 = false 1 = true	1	0,1	0,1
`byte`	0...255	2 (4)	‘00’,’FF’,’ff’,’7a’, ’A7’	‘00’,’FF’,’ff’, ’7a’,’A7’
`string`	-	<512	”abc\x0d\x0a”	”abc\x0d\x0a”
`double`	±2.225E-308... ±1.797E+308	17+3	1, 1.0, 1.0e4, -0.1e-07, -2	-0.1234567+e67
`long`	$(-2^{31})...(2^{31}-1)$	11	0x7FFFFFFF, -54321	15, -154836, 900000
`short`	-32768...32767	6	0, -1, -32700, 45, 56, 0x45e, 0X3AA	0, -1, -32700, 45, 56
`unsigned long`	$0...(2^{32}-1)$	10	0xFFFFFFFF	0, 1, 3400065, 95735
`unsigned short`	0...65535	5	0, 1, 34000, 65, 65535, 0x3a, 0x00, 0xFFFF	0, 1, 34000, 65, 65535

Table 3-1: Communication Parameter Types

Note: Bytes are always represented in two-character hexadecimal notation. Hexadecimal notation can use upper- or lower-case representation: 0..9 + [a .. f | A .. F].
Characters sent within a string which do not fall within the ASCII character range 0x20 to 0x7E (32 to 126 decimal) are sent using an adapted byte notation - e.g. ”\x9A”, where \x (or \X) introduces a byte value in hexadecimal notation.
Types of integer (short, unsigned short, long, unsigned long) can also be represented in hexadecimal notation, introduced by 0x or 0X.

The following rules are for generating/interpreting values with a type different from the base types and aggregated data types:

Numerical and string data type

The numerical data types correspond to the C-parameters in value, range and precision as close as possible. If no identical data type is available then the next best one will be taken. Character and string will be replaced by the `string` data type.

Enumerations

If the corresponding C-parameter is an enumeration data type, then the enumeration value of the ASCII parameter is equal to the implicit value of the declaration of the C-data type. For clarification, we will give always the name and the associated value in the description of an enumeration data type.

Structures

Structure data types will be converted into a comma separated list of elements. One element's representation conforms to the data type representation of its base type. If an element itself is a structure then depth first conversion will take place. If this rule does not apply then the types and their ASCII parameters are described explicitly.

Arrays

An array will be converted into a comma-separated list of elements. One element's representation conforms to the data type representation of its base type.

Example for Enumeration Data Types and Structures

The following example gives a typical data type declaration and the corresponding procedure declaration used in this manual for `TMC_GetSimpleMea` from the subsystem Theodolite Measurement and Calculation:

Constants and Types

```
typedef long SYSTIME;

struct TMC_HZ_V_ANG
{
  double dHz;
  double dV;
}
```

```
enum TMC_INCLINE_PRG
{
  TMC_MEA_INC,              // encoded as 0
  TMC_AUTO_INC,             //            1
  TMC_PLANE_INC             //            2
}
```

C-Declaration

```
TMC_GetSimpleMea(SYSTIME          WaitTime,
                 TMC_HZ_V_ANG     &OnlyAngle,
                 double           &dSlopeDistance,
                 TMC_INCLINE_PRG  Mode)
```

ASCII-Request

`%R1Q,2108:`*WaitTime[long],Mode[long]*

ASCII-Response

`%R1P,0,0:`*RC,Hz[double],V[double],dSlopeDistance[double]*

Please, notice that the RPC has two input and two output parameters. Anytime a request must encode and send input and in/out parameters only and a reply must encode and send in/out and output parameters only!

Note: Unnecessary parameters must not be sent.
Although the enclosed header file `com_pub.hpp` denotes default values for certain function parameters they will not be supported. Hence, they have to be sent.

The ASCII Request to call this RPC with the value for `WaitTime` = 1000 and the inclination measure mode `TMC_AUTO_INC` has the following form (note that the value 1 is used for the `Mode` parameter because the counting of enumeration data types start at 0):

```
%R1Q,2108:1000,1^m
```

A possible reply can be as follows:

```
%R1P,0,0:0,0.9973260431694,1.613443448007,1.3581^m
```

Where the second and third value after the colon corresponds to the `dHz` and `dV` parts of the structure `TMC_HZ_V_ANG` and the fourth value corresponds to the variable `dSlopeDistance`. (Note that the first value after the ':' is not a parameter but the return code value of the RPC).

3.1.2 ASCII Protocol Program Example

For getting a feeling of how requests and replies are build up and work see also the provided `geocom.trm` file in the samples directory. Please refer to Appendix **C-1 Settings for Terminal Emulator** for further information.

3.1.3 Modes of Operation Concerning Communication

Section 3.5 - TPS1000 Instrument Modes of Operation - explains the different modes of operation of GeoCOM concerning communication. Similar to that the following is valid for the ASCII protocol.

Since the client has to remind which mode is active, no support can be given from the TPS1000 instrument. The only way to distinguish between modes is to remind the actions an application has initiated and their resulting replies. So far no other possibility exists to determine the current mode.

To switch on the instrument a single character is sufficient. It is recommended to ignore the subsequent reply (one or two lines). Please note, that if the autoexec mechanism ([menu] CONF – [item] Autoexec-application) is enabled then the instrument will not switch into Remote mode, but will start the autoexec application instead.

When turning into local mode the TPS1000 instrument sends the “sign-on” message:

`“%N1,0,255,,0%T0,0,0,:%R1P,0,0:0”`.

If the “sign-off” message is enabled (see `COM_EnableSignOff`) then the following message will be sent if the instrument goes into sleep mode:

`“%N1,0,255,,0%T0,0,0,:%R1P,1,0:0,1”`,

and the following message will be sent if the instrument shuts down:

`“%N1,0,255,,0%T0,0,0,:%R1P,1,0:0,0”`.

Please notice that these two messages are different in the last character.

3.2 C/C++ - PROGRAMMING

Programming in C/C++ is based on the well-known DLL concept, defined by Microsoft Corp. To compile a project successfully first you have to include the file `com_pub.hpp`, which defines all necessary constants, data types and function prototypes. Second `geocom.lib` has to be included in the project, which enables

the linker to resolve the DLL exported functions. To operate successfully the `geocom.dll` file must be accessible for the operating system, hence it must be located in a directory which the operating system looks up for the requested DLL file.

Project Options	**Geocom.lib**	**Geocom32.lib**
Structure byte-alignment	1 byte	4 bytes
Memory model	Large	N/A
Special #defines (if not using MFC)	STRICT	STRICT

3.2.1 Data Types in C/C++

Since the main programming language of implementation of TPS1000 instruments Firmware is C/C++ all data types are initially defined in C/C++. Therefore, no conversion of values or data types is necessary.

3.2.2 Implementation Differences - 16/32 Bit

The implementation of 16 and 32 Bit interfaces are identical. Please note that instead of `geocom.lib` and `geocom.dll` the 32 Bit versions have to be used: namely `geocom32.lib` and `geocom32.dll`.

3.2.3 Basic GeoCOM Application Frame for C/C++

A C/C++ GeoCOM application consists at least of the following parts:

- Initialise GeoCOM
- Open a connection to the server
- One or more GeoCOM RPC’s
- Close the active connection to the server
- Finalise GeoCOM

A sample implementation of above points could be:

```
// include standard system headers
#include "com_pub.hpp"
// include application headers
```

```
#define RETRIES_1 1

RC_TYPE   RetCode;
BOOLE     bOpenAndRunning = FALSE;

// initialize GeoCOM
RetCode = COM_Init();
if (RetCode == RC_OK)
{
  // open a connection to the TPS1000 instrument
  RetCode = COM_OpenConnection (  COM_1, COM_BAUD_19200,
                                  RETRIES_1);
  if (RetCode == RC_OK)
  {
    bOpenAndRunning = TRUE;
  }
}

// optionally set up other comm. parameters here

if (RetCode == RC_OK)
{
  // -- functionality of the application --
  // here we just test if communication is up
  RetCode = COM_NullProc();
  if (RetCode != RC_OK)
  {
    // handle error
  }
}

// close channel
if (bOpenAndRunning)
{
  RetCode = COM_CloseConnection ();
  if (RetCode != RC_OK)
  {
    // handle error
  }
}

// anytime finalize and reset GeoCOM
RetCode = COM_End();
if (RetCode != RC_OK)
{
```

```
  // handle error
}
```

3.2.4 C/C++ Development System Support

GeoCOM system files have been developed using Microsoft Visual C/C++ 1.52 and 4.2. Although these development environments were the basis for the current GeoCOM implementation, it has been emphasised that it is independent of them, hence other development environments can be used too. But please notice that it has not been tested thoroughly so far.

3.2.5 Programming Hints

Order of Include Statements

Since GeoCOM redefines `TRUE`, `FALSE` and `NULL` we recommend the following include order:

1. Include system headers like `stdio.h` or `stdafx.hpp`
2. Include `com_pub.hpp`
3. Include the current project headers

BOOLE Definition

GeoCOM defines its own Boolean type as an enumeration type of `FALSE` and `TRUE`. It is called `BOOLE`. With one exception, this does not produce any problems. Only if a `BOOL` type value will be assigned to a `BOOLE` type variable or parameter the compiler (MS-VisualC/C++) generates an error. To solve this problem the expression, which will be assigned to, has to be converted by a `CAST` statement to `BOOLE`.

3.3 VBA – PROGRAMMING

Similar to C/C++ programming the programming of VBA is based on the DLL concept. To enable access to GeoCOM the special module `stubs_p.bas` has to be included in the project. `stubs_p.bas` includes all constants, data types and function prototypes, which are available in GeoCOM.

3.3.1 Data Types in VBA - General rules for derivation

This subsection gives a summary of general derivation rules VBA-parameters from C-data types. Basically the C/C++ - data types are given in a C/C++ notation before they are used in a RPC-description.

If the appearance of a VBA data type does not follow the general rules then they are described explicitly.

In general, the following rules can be applied:

Numerical data type

The numerical data types correspond to the C/C++-parameters in value and range as close as possible. If it cannot be replaced directly then the best possible replacement will be taken.

String data type

Character and string types are replaced by `string` data types. Since string data types of C/C++ and VBA are not directly interchangeable, the programmer has to take certain care of the necessary pre- and post-processing of variables of this data type. Please refer to the example below.

Enumeration data type

Conceptually VBA does not have enumeration data types. Therefore, `Integer` (`Long`) data types will be used instead. The enumeration values will be defined by constants. Using the numerical value is also valid. Notice that some of the enumeration values are reserved words in VBA. That is why we had to define different identifiers. Enumerated return values are numerical values and correspond to the position of the enumeration value in the C/C++-definition. For clarification, also the numerical values are given in the description of an enumeration data type.

Note: The 16 Bit version uses `Integer` and the 32 Bit version uses `Long` as the replacement for C/C++ enumeration type.

Structures and Arrays

They are defined as in C/C++.

Example for Enumeration Data Types and Structures

The following example gives the data type declaration and the procedure declaration usually used in this manual for an example procedure

(TMC_GetSimpleMea from the subsystem Theodolite Measurement and Calculation):

VBA-Declaration (16 Bit)

```
VB_TMC_GetSimpleMea(
          WaitTime        As Long,
          OnlyAngle       As TMC_HZ_V_ANG,
          SlopeDistance   As Double,
          Mode             As Integer)
```

In the file stubs_p.bas file the corresponding items are defined:

```
Global Const TMC_MEA_INC = 0
Global Const TMC_AUTO_INC = 1
Global Const TMC_PLANE_INC = 2

Type TMC_HZ_V_ANG
  dHz  As Double
  dV   As Double
End Type
```

Obviously all enumeration values are encoded as global constants. The VBA structure definition equals to the C structure definition. A valid procedure call would be:

```
Dim WaitTime       As Long
Dim OnlyAngle      As TMC_HZ_V_ANG
Dim SlopeDistance  As Double

WaitTime = 1000

VB_TMC_GetSimpleMea( WaitTime,
                     OnlyAngle,
                     SlopeDistance,
                     TMC_AUTO_INC)
```

3.3.2 Implementation Differences - 16/32 Bit

The implementation of 16 and 32 Bit interfaces are almost identical. Please note that instead of geocom.dll the 32 Bit version has to be used: namely geocom32.dll. Also stubs32p.bas has to be used instead of stubs_p.bas.

Furthermore, the implementation of enumeration data types is different between VBA16 and VBA32. In the former case `Integer` and in the latter `Long` will be used as their replacements. Please refer to 3.3.5 - Programming Hints also.

3.3.3 Basic GeoCOM Application Frame for VBA

Like in section 3.2.3 - Basic GeoCOM Application Frame for C/C++ - a VBA GeoCOM application consists at least of the following parts:

- Initialise GeoCOM
- Open a connection to the server
- One or more GeoCOM RPC's
- Close the active connection to the server
- Finalise GeoCOM

A sample implementation of above points could be:

```
CONST RETRIES_1 = 1
DIM RetCode As Integer
DIM bOpenAndRunning as Integer

' initialize GeoCOM
bOpenAndRunning = False
RetCode = VB_COM_Init()
If (RetCode = RC_OK) Then
  ' open a channel to the TPS1000 instrument
  RetCode = VB_COM_OpenConnection(COM_1, COM_BAUD_19200,
                                  RETRIES_1)
  If (RetCode = RC_OK) Then
    bOpenAndRunning = True
  End If
End If
' optionally set up other comm. parameters here

If (RetCode = RC_OK)  Then
  ' functionality of the application
  ' we just test if communication is up
  RetCode = VB_COM_NullProc()
  If (RetCode <> RC_OK) Then
    ' handle error
  End If
End If
```

```
If (bOpenAndRunning) Then
  ' close channel
  RetCode = VB_COM_CloseConnection ()
  If (RetCode <> RC_OK) Then
    ' handle error
  End If
End If

' finalize and reset GeoCOM
RetCode = VB_COM_End()
If (RetCode <> RC_OK) Then
  ' handle error
End If
```

3.3.4 VBA Development System Support

This interface has been written for Microsoft Visual Basic for Applications 4.0 and higher only. Hence, no other development environment will be supported.

3.3.5 Programming Hints

Combining 16/32 Bit Targets

To handle 16 and 32 Bit environments we recommend including both `stubs_p.bas` and `stubs32p.bas` into your project. As already mentioned originally defined enumeration values will be defined as either `Integer` or `Long`. For instance, see below for the code fragment.

```
#If Win16 Then
  Dim nPort As Integer
#Else
  Dim nPort As Long
#End If
Dim RetCode As Integer

RetCode = VB_COM_Init()
If RetCode = RC_OK Then
  ' port is set outside of this procedure
  If port = 0 Then
    nPort = COM_1
  Else
```

```
    nPort = COM_2
  End If
  RetCode = VB_COM_OpenConnection(nPort,
                                  COM_BAUD_9600, 1)
End If

...
```

Output Parameters of String Data Type

The internal representation of strings is not directly compatible between C/C++ and VBA. Therefore the one has to pre- and post-process such an output parameter. In the following example, we know that the output parameter will be less than 255 characters in length from the description of the RPC.

```
Dim s As String

' initialise string
s = Space(255)
Call VB_COM_GetErrorText(RC_IVPARAM, s)
' trim string, justify string length
s = Trim$(s)
```

Note: Incorrectly handled string output parameters may lead to severe runtime problems.

3.4 UNITS OF VALUES

All parameters are based on the SI unit definition, if not explicitly indicated differently. The SI units, and their derivatives, used are:

Abbreviation	Unit	Description
M	(Meters)	for lengths, co-ordinates, ...
Rad	(Radians)	for angles
Sec	(Seconds)	for time
Hpa	(Hekto Pascal)	for pressure
C	(Celsius)	for temperature

Table 3-2: SI Units

3.5 TPS1000 INSTRUMENT MODES OF OPERATION

In respect to communication, the TPS1000 instrument knows several states in which it reacts differently. The main state for GeoCOM is online state or mode. There it is possible to use all RPC's, which are described in this manual. Especially we will describe the possibilities of changing the state by the built-in RPC's. For the ASCII protocol refer to section 3.1.3 - Modes of Operation Concerning Communication.

The possible states can be described as follows:

Off The instrument is switched off and can be switched on and put into online mode by using `COM_SwitchOnTPS`.

Local The instrument is in local mode. GeoCOM is not active hence, RPC's cannot be used. Switch into online mode with menu EXTRA on the instrument.

Online Also called Remote mode. The instrument accepts RPC's. `COM_Local` can be used to switch into local mode. `COM_SwitchOffTPS` will switch off the instrument or put it into sleep mode.

RCS The instrument accepts Remote Control sequences. This is not subject of this manual and will be described elsewhere.

GSI/Meas In this mode the user can measure coordinates and save the values onto the PC-Card. GSI commands will be accepted in this mode. Since this is not subject of this manual this mode will not described here in more detail.

Sleep Either because of reaching the time out or by using the function `COM_SwitchOff(COM_TPS_SLEEP)` this state has been reached when starting from online mode. Only if the previous mode was online mode it can be switched back to it with `COM_SwitchOnTPS(COM_TPS_REMOTE)`.

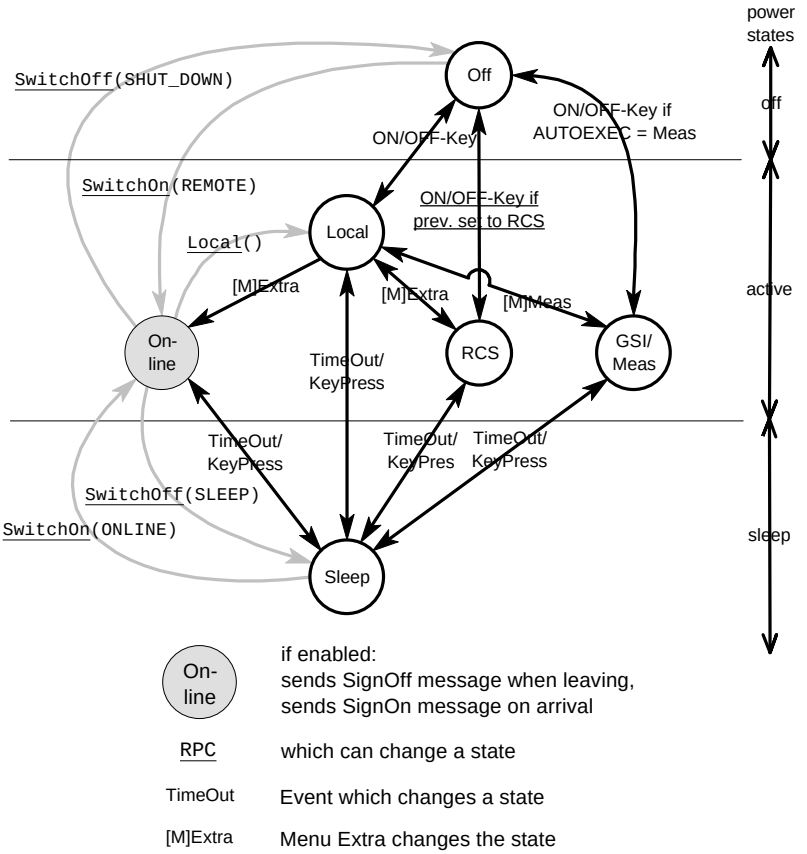
3.5.1 Getting Mode of Operation Concerning Communication

This is available only when the application uses the function call protocol. Hence a DLL is used to automate RPC calls. Earlier implementations do not support logging of the current state. That is why the current implementation does not log the current state per default. But with Release 2.20 and higher logging can be switched on by using `COM_EnableSignOff(TRUE)`. After enabling state changes

to and from online mode will be logged and can be requested with `COM_GetTPSState`.

Note: The mode can be determined only if GeoCOM is active and the sign-off message is activated. In any other situation, `COM_GetTPSState` will yield the state `COM_TPS_UNKNOWN`.

The following picture shows the dependencies graphically. Instead of the correctly defined identifiers, shortcuts have been used.



Picture 3-1 TPS1000 modes of operation in respect to communication

3.6 COMMON COMMUNICATION ERRORS

GeoCOM is based on calling functions remotely. Because of the additional communication layer the set of return codes increases with return codes based on communication errors. Since all of these codes may be returned by any RPC we will explain them here and omit them in the descriptions of the RPC's.

Return Code	Val	Description
RC_OK	0	Successful termination, implies also no communication error.
RC_COM_CANT_ENCODE	3073	Can't encode arguments in client. Returned by the client to the calling application directly, i.e. without anything being sent to the transport layer and beyond.
RC_COM_CANT_DECODE	3074	Can't decode results in client. Once an RPC has been sent to the server and a reply has been sent back, this return code states that the encoded reply could not be decoded in the client. This is usually the result of using different versions of GeoCOM on client and server.
RC_COM_CANT_SEND	3075	Failure in sending calls. If the resources at the transmitting port have been allocated previously, i.e. GeoCOM does not have exclusive rights to the port, or if the exception or similar routine has experienced a failure, this error code is returned.
RC_COM_CANT_RECV	3076	Failure in receiving result. A failure has occurred during reception of a packet at the data link layer. This could be due to incorrect parameter settings or noise on the line, etc..

Return Code	Val	Description
RC_COM_TIMEDOUT	3077	Call timed out. The client has sent an RPC to the server but it has not replied within the current time-out period as set for the current transaction. This could be because: the server has not received the request; the server has taken too long to execute the request; the client has not received the reply; the communication line (physical layer is no longer there; or, the time-out is too short (especially true when communicating over noisy or radio links at low baud rates).
RC_COM_WRONG_FORMAT	3078	The request and receive formats are different. Something got mixed up along the way or the application tried to send using a format which has not been implemented on both client and server.
RC_COM_VER_MISMATCH	3079	RPC protocol mismatch error. An RPC protocol has been requested which does not exist. This error will indicate incompatible client and server protocols.
RC_COM_CANT_DECODE_REQ	3080	Can't decode request in server. If the client sends the server an RPC but one which cannot be decoded in the server, the server replies with this error. It could be that the GeoCOM versions running on the client and server are different or the packet was not correctly sent over a noisy or unreliable line.
RC_COM_PROC_UNAVAIL	3081	The requested procedure is unavailable in the server. An attempt has been made to call an RPC, which does not exist. This is usually caused when calling RPC's which have been inserted, appended, deleted, or altered between the differing versions of GeoCOM on client and server. To be on the safe side, always use the same GeoCOM version whenever possible on both sides.

Return Code	Val	Description
RC_COM_CANT_ENCODE_REP	3082	Can't encode reply in server. The server has attempted to encode the reply but has failed. This can be caused by the calling procedure trying to pass too much data back to the client and in so doing has exceeded the maximum packet length.
RC_COM_SYSTEM_ERR	3083	Communication hardware error
RC_COM_FAILED	3085	Mess into communication itself. Should be OK once the node has been recycled, i.e. powered-down and -up again.
RC_COM_NO_BINARY	3086	Unknown protocol. An unknown (or not yet supported) Transport or Network protocol has been used. Could appear when using differing GeoCOM versions on client and server. Not used up to 2.10
RC_COM_INTR	3087	Call interrupted. Something has happened outside of the scope of GeoCOM, which has forced the current RPC to abort itself.
RC_COM_REQUIRES_8DBITS	3090	This error indicates desired protocol requires 8 data bits (only Version 2.20 or later)
RC_COM_TR_ID_MISMATCH	3093	Request and reply transaction ids do not match. Somewhere along the line a packet (usually a reply) has been lost or delayed. GeoCOM tries to bring everything back to order but if this error continues during the session it may be wise to inspect the line and, at least, to restart the session. The immediately following RPC may be lost.
RC_COM_NOT_GEOCOM	3094	Parse failed; data package not recognised as GeoCOM communication package
RC_COM_UNKNOWN_PORT	3095	Tried to access an unknown hardware port. The application has not taken the physical resources of the machine on which it is running into account.

Return Code	Val	Description
RC_COM_OVERRUN	3100	Overruns during receive. A packet has been received which has exceeded the maximum packet length. It will be discarded! This can be caused by a noisy line during GeoCOM Binary format transmissions.
RC_COM_SRVR_RX_ CHECKSUM_ERROR	3101	Checksum received at server is wrong. The checksum belonging to the current packet is wrong - no attempt is made at decoding the packet.
RC_COM_CLNT_RX_ CHECKSUM_ERROR	3102	Checksum received at client is wrong. The checksum belonging to the current packet is wrong - no attempt is made at decoding the packet.
RC_COM_PORT_NOT_ AVAILABLE	3103	COM port not available. This can be caused by attempting to open a port for unique use by GeoCOM, which has already been allocated to another application.
RC_COM_PORT_NOT_ OPEN	3104	COM port not opened / initialised. The application has attempted to use a COM port to which it has no unique rights.
RC_COM_NO_PARTNER	3105	No communications partner on other end. The connection to the partner could not be made or has been lost. Check that the line is there and try again.
RC_COM_ERO_NOT_ STARTED	3106	The client, after calling an ERO has decided not to confirm the start of the ERO and has instead called another RPC.

Return Code	Val	Description
RC_COM_CONS_REQ	3107	Attention to send consecutive requests. The application has attempted to send another request before it has received a reply to its original request. Although GeoCOM does not return control to the app until a reply is received, this error is still possible with event-driven applications, i.e., the user pushing a button yields control back to the application code which can then call GeoCOM again.
RC_COM_SRVR_IS_SLEEPING	3108	TPS has gone to sleep. Wait and try again.
RC_COM_SRVR_IS_OFF	3109	TPS has shut down. Wait and try again

4 REMARKS ON THE DESCRIPTION

This chapter contains some remarks on the description of RPC's and on the structure of the descriptions.

4.1 STRUCTURE OF DESCRIPTIONS

The whole reference part is subdivided into sections. Each section contains descriptions of a set of functions, which build up a subsystem. A subsystem gathers all functions, which are related to a specific functionality of a TPS1000 instrument, e.g. MOT describes all functions, which relate to motorization. Each subsystem is subdivided into the descriptions of RPC's.

4.1.1 Structure of a Subsystem

A subsystem consists of the following parts:

1. **Usage**
   This part gives some hints about the usage of the subsystem and general information of its functionality.
2. **Constants and Types**
   All subsystem specific constants and data types are listed here. Also their meanings are described if they are not obvious.
3. **Functions**
   All RPC's of this subsystems are listed here and described in detail.

Note: To reduce redundancy the VB declarations of data types and constants have been omitted. Please refer to chapter 3.3 to get more information about this subject.

4.1.2 Structure of a RPC Description

One RPC description contains the following parts:

Title

Contains the name of the RPC and a short description of the function.

C-Declaration

Contains the C declaration of the function (excluding the return type).

VB-Declaration

Declares the function in VB (excluding the return type).

ASCII-Request

Describes the composition, inclusive the base types, of the ASCII request.

ASCII-Reply

Describes the composition, inclusive the base types, of the corresponding request.

Remarks

Gives additional information on the usage and possible side effects of the function.

Parameters

Explains the parameters, their data types and their meaning.

Return-Codes

Gives the meaning of the return codes related to this RPC. General and communication return codes will be omitted in explanations. They are explained in section 3.6.

See Also

Cross references shows other RPC's which relate to this one.

Example

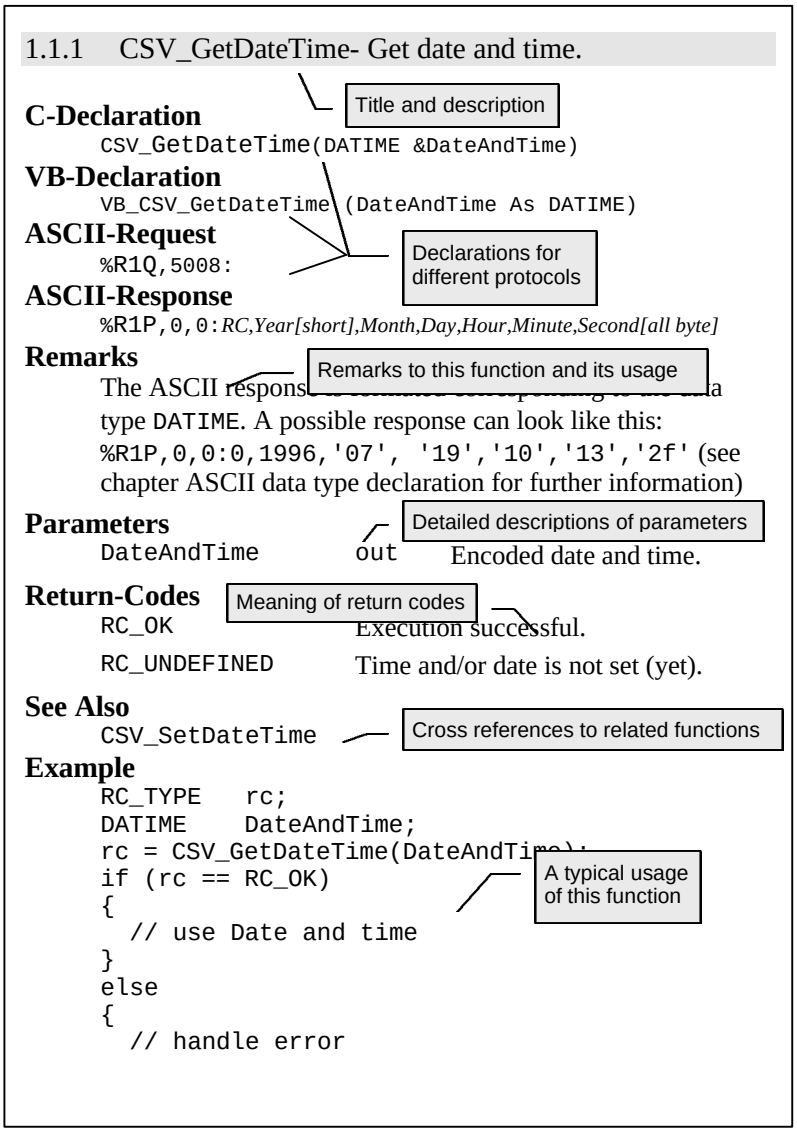
Gives an example of how this RPC could be used.

Note: To reduce redundancy the return type has been omitted from the C- and VB-declarations of the RPC's.

ASCII-Request and Reply do not explain the whole data structures. Instead the corresponding base types will be given. Please refer to chapter 2.2 to get more information on this topic.

Also because of redundancy the necessary CR/LF at the end has been omitted from ASCII-Request and Reply.

4.1.3 Sample of a RPC Description



5 COMMUNICATION SETTINGS

This subsystem provides functions which influences GeoCOM as a whole and functions, which relate to the client side only.

If a function influences the client side only then there is no ASCII request defined.

5.1 CONSTANTS AND TYPES

Serial Port Selector

This enumeration type denotes the hardware serial port.

```
enum COM_PORT
{
        COM_1   = 0,            // port 1
        COM_2   = 1,            // port 2
        COM_3   = 2,            // port 3
        COM_4   = 3             // port 4
  };
```

Transmission Data Format

This value tells if the transmission takes place in a readable ASCII data format or in a data size optimised binary data format.

```
enum COM_FORMAT
{
        COM_ASCII  = 0,         // Force ASCII comm.
        COM_BINARY = 1          // Enable binary comm.
};
```

Baud Rate

```
enum COM_BAUD_RATE
{
        COM_BAUD_38400 = 0,
        COM_BAUD_19200 = 1,     // default baud rate
        COM_BAUD_9600  = 2,
        COM_BAUD_4800  = 3,
        COM_BAUD_2400  = 4
};
```

TPS1000 Operation Status

```
enum COM_TPS_STATUS
{
        COM_TPS_OFF         = 0,    // switched off
        COM_TPS_SLEEPING    = 1,    // sleep mode
        COM_TPS_READY       = 2,    // online mode
        COM_TPS_UNKNOWN     = 3     // unknown mode, because mode isn't
                                    // supported or signoff is disabled
};
```

MS-Windows Data Types

One of the described functions uses the predefined type `HWND` of MS-Windows. Please refer to the documentation of MS-Windows development environment for this data type.

Note:	`HWND` depends on whether the pre-processor symbol `STRICT` is defined. When MFC libraries are used, `STRICT` is automatically defined. Otherwise the user must `#define STRICT` or he will get unresolved externals.

5.2 GENERAL GEOCOM FUNCTIONS

5.2.1 COM_GetDoublePrecision - Get Double Precision Setting

C-Declaration

```
COM_GetDoublePrecision( short &nDigits )
```

VB-Declaration

```
VB_COM_GetDoublePrecision( nDigits As Integer )
```

ASCII-Request

```
%R1Q,108:
```

ASCII-Response

```
%R1P,0,0:RC, nDigits[short]
```

Remarks

This function returns the precision – number of digits to the right of the decimal point – when double floating-point values are transmitted. The

usage of this function is only meaningful if the communication is set to ASCII transmission mode. Precision is equal in both transmission directions. In the case of an ASCII request, the precision of the server side will be returned.

Parameters

NDigits	Out	Number of digits to the right of the decimal point.

Return Codes

RC_OK	On successful completion.

See Also

COM_SetDoublePrecision

Example

```
RC_TYPE         rc;
short           nDigits, nOldDigits;
TMC_HEIGT       height;

 (void) COM_GetDoublePrecision(nOldDigits);
rc = COM_SetDoublePrecision(nDigits);

// nDigits > 15, nDigits < 0 -> RC_IVPARAM
if (rc == RC_IVPARAM)
{
  rc = COM_SetDoublePrecision(7);
}

// measure height of reflector ...

// the result is precisely calculated and
// returned with nDigits to the right of the
// decimal point

(void) TMC_GetHeight(height);     // ignore return code
print(„height: %d\n“, height.dHr);

// reset server accuracy to the old value
rc = COM_SetDoublePrecision(nOldDigits);

// no error handling, because nOldDigits must be valid
```

5.2.2 COM_SetDoublePrecision - Set Double Precision Setting

C-Declaration

```
COM_SetDoublePrecision( short nDigits )
```

VB-Declaration

```
VB_COM_SetDoublePrecision( ByVal nDigits As Integer )
```

ASCII-Request

`%R1Q,107:`*nDigits[short]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function sets the precision – number of digits to the right of the decimal – when double floating-point values are transmitted. The TPS' system software always calculates with highest possible precision. The default precision is fifteen digits. However, if this precision is not needed then transmission of double data (ASCII transmission) can be speeded up by choosing a lower precision. Especially when many double values are transmitted this may enhance the operational speed. The usage of this function is only meaningful if the communication is set to ASCII transmission mode. In the case of an ASCII request, the precision of the server side will be set. Notice that trailing Zeros will not be sent by the server and values may be rounded. E.g. if precision is set to 3 and the exact value is 1.99975 the resulting value will be 2.0

Note:	With this function one can decrease the accuracy of the delivered values.

Parameters

`nDigits`	`In`	Number of digits right to the comma.

Return Codes

`RC_OK`	On successful completion.
`RC_IVPARAM`	`0 > nDigits > 15`

See Also

`COM_GetDoublePrecision`

Example

see `COM_GetDoublePrecision`

5.3 CLIENT SPECIFIC GEOCOM FUNCTIONS

The following functions are not applicable to the ASCII protocol, because these functions influence the behaviour of the client application only.

5.3.1 COM_Init - Initialize GeoCOM

C-Declaration

```
COM_Init( void )
```

VB-Declaration

```
VB_COM_Init()
```

ASCII-Request

-

ASCII-Response

-

Remarks

`COM_Init` has to be called to initialise internal buffers and variables. It does not change the TPS' state.

Note: No other GeoCOM function can be called successfully without having initialised GeoCOM before.

Parameters

-

Return Codes

`RC_OK`	On successful completion.

See Also

`COM_End`

Example

See chapter **C-2 Program Frames** for an example program frame

5.3.2 COM_End - Quit GeoCOM

C-Declaration

```
COM_End( void )
```

VB-Declaration

```
VB_COM_End()
```

ASCII-Request

-

ASCII-Response

-

Remarks

`COM_End` has to be called to finish up all open GeoCOM transactions. It closes an open port and does whatever is necessary to shutdown GeoCOM. The TPS' state will not be changed.

Parameters

-

Return Codes

`RC_OK`	On successful completion.

See Also

`COM_Init`

Example

see `COM_Init`

5.3.3 COM_OpenConnection - Open a Port for Communication

C-Declaration

```
COM_OpenConnection( COM_PORT        ePort,
                    COM_BAUD_RATE  &eRate,
                    Short           nRetries )
```

VB-Declaration

```
VB_COM_OpenConnection( ByVal Port     As Integer,
                       ByVal Baud     As Integer,
                       ByVal Retries  As Integer )
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function opens a PC serial port and attempts to detect a theodolite based on the given baud rate. If a TPS is well connected to the PC then GeoCOM tries to establish a connection to it.

If no connection could be established and the connection dialog flag is set to TRUE then a dialog appears which asks the user if all possible baud rate settings should be tried. See also `COM_SetConnDlgFlag`. If the flag is cleared, all possible settings will be tried automatically without notification of the user.

To be successful the TPS must be in online mode.

If the TPS is switched off it will be switched on remotely and set into online mode automatically.

The default transmission data format is set to `COM_BINARY`, if the TPS Release is equal or higher than 2.20.

This function will fail if the TPS is in local-mode or if the serial-port is locked or in use. It will also fail if no TPS is connected to the serial port.

If the call cannot be finished successfully then the port will be freed and closed.

The successful completion of this may take more than a minute, if all possible settings have to be tried out.

`nRetries` denotes the number of retries of subsequent RPCs if the first request has not been fulfilled successfully. Especially for radio data links this is of interest if the link is not reliable. We recommend not using a value higher than two since this may slow down communication significantly.

Note: In the current implementation, GeoCOM does not support two open connections at the same time. A second attempt to open a second port at once will be denied by GeoCOM.

Parameters - C-Declaration

`EPort`	`In`	Serial port.

eBaud	InOut	Baud rate.
nRetries	In	Number of retries.

Return Codes

RC_OK	On successful completion.
RC_COM_PORT_NOT_AVAILABLE	Port is in use or does not exist
RC_COM_NO_PARTNER	GeoCOM failed to detect a TPS.
RC_IVPARAM	Illegal parameter.

See Also

COM_CloseConnection
COM_SetConnDlgFlag

Example

see COM_Init

5.3.4 COM_CloseConnection - Close the Open Port

C-Declaration

```
COM_CloseConnection( void )
```

VB-Declaration

```
VB_COM_CloseConnection( )
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function closes the (current) open port and releases an established connection. It will not change the TPS' state.

Parameters

-

Return Codes

RC_OK	On successful completion.

See Also

COM_OpenConnection

Example

See chapter **C-2 Program Frames** for an example program frame

5.3.5 COM_GetBaudRate – Get Current Baud Rate

C-Declaration

```
COM_GetBaudRate ( COM_BAUD_RATE &eRate )
```

VB-Declaration

```
VB_COM_GetBautRate( eRate As Integer )
```

ASCII-Request

-

ASCII-Response

-

Remarks

Get the current baud rate of the serial line. It should be the setting of both client and server. In ASCII protocol, this RPC is not available.

Parameters

`eRate`	`Out`	Baud rate of serial line.

Return Codes

`RC_OK`	On successful completion.

See Also

`COM_SetBaudRate`

Example

```
void main()
{
     RC_TYPE              rc;
     COM_BAUD_RATE        eRate;

     // init GeoCOM
     ...

     // get baud rate of active connection
       rc = COM_GetBaudRate(eRate);
       if (rc != RC_OK)
       {
         COM_ViewError(rc, "Setup baud rate");
```

```
            }
            else
            {
            printf("Baudrate is %d Baud = " );
            switch (eRate )
            {
              case COM_BAUD_38400:
                printf("38400\n");
                break ;
              case COM_BAUD_19200:
                printf("19200\n");
                break ;
              case COM_BAUD_9600:
                printf("9600\n ");
                break ;
              case COM_BAUD_4800:
                printf("4800\n ");
                break ;
              case COM_BAUD_2400:
                printf("2400\n ");
                break ;
              default:
                printf("illegal\n ");
                break ;
            }
          }

          ...
          // shutdown GeoCOM

      }     // end of main
```

5.3.6 COM_SetBaudRate - Set Baud Rate

C-Declaration

```
COM_SetBaudRate( COM_BAUD_RATE  eRate )
```

VB-Declaration

```
VB_COM_SetBaudRate( ByVal eRate As Integer )
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function sets the baud rate of the serial line, hence on both client and server side. A port must have been opened successfully with `COM_OpenConnection`.

Parameters

`eRate`	`In`	Baud rate.

Return Codes

`RC_OK`	On successful completion.

See Also

`COM_GetBaudRate`

Example

```
RC_TYPE         rc;


// set baud rate
rc = COM_SetBaudRate(COM_BAUD_9600);
if (rc == RC_IVPARAM)
{
  // handle errors
}
```

5.3.7 COM_GetTimeOut - Get Current Timeout Value

C-Declaration

```
COM_GetTimeOut( short &nTimeOut )
```

VB-Declaration

```
VB_COM_GetTimeOut( nTimeOut As Integer )
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function retrieves the current timeout value for a request in seconds. The timeout value is the delay GeoCOM will wait for completion before it signals an error to the calling application.

Parameters

`nTimeOut`	`Out`	Timeout value in seconds, default value is 5 sec.

Return Codes

`RC_OK`	On successful completion.

See Also

`COM_SetTimeOut`

Example

```
RC_TYPE   rc;
shortnTimeOut;

COM_GetTimeOut(nTimeOut);

if (nTimeOut <= 5)
{
  COM_SetTimeOut(7);
}
```

5.3.8 COM_SetTimeOut - Set Current Timeout Value

C-Declaration

```
COM_SetTimeOut( short nTimeOut )
```

VB-Declaration

```
VB_COM_SetTimeOut( nTimeOut As Integer )
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function sets the current timeout value in seconds. The timeout value is the delay GeoCOM will wait for completion of the last RPC before it signals an error to the calling application.

A zero timeout value indicates no wait. This can be used for polling the input queue. But be aware of that this will yield into a `RC_COM_TIMEDOUT` return code.

Note: A negative timeout value indicates an infinite waiting period and may block the client application.

Parameters

`nTimeOut`	`In`	timeout value in seconds

Return Codes

`RC_OK`	On successful completion.

See Also

`COM_GetTimeOut`

Example

see `COM_GetTimeOut`

5.3.9 COM_GetComFormat - Get Transmission Data Format

C-Declaration

```
COM_GetComFormat( COM_FORMAT &eComFormat )
```

VB-Declaration

```
VB_COM_GetComFormat( eComFormat As Integer )
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function gets the actual transmission data format. GeoCOM uses `COM_BINARY` as a default with Firmware Releases higher than 2.20. But if the TPS' Firmware does not support binary transmission data format then ASCII data format will be used instead.

Parameters

`eComFormat`	`Out`	`COM_ASCII` or `COM_BINARY`

Return Codes

RC_OK	On successful completion.

See Also

COM_SetComFormat

Example

```
RC_TYPE          rc;
COM_FORMAT    eComFormat;

COM_GetComFormat(eComFormat);
if (eComFormat == COM_ASCII)
{
   printf(„ASCII mode in use.\n“);
}
else
{
   printf(„BINARY mode in use.\n“);
}
```

5.3.10 COM_SetComFormat - Set Transmission Data Format

C-Declaration

```
COM_SetComFormat( COM_FORMAT eComFormat )
```

VB-Declaration

```
VB_COM_SetComFormat( ByVal eComFormat As Integer )
```

ASCII-Request

-

ASCII-Response

-

Remarks

GeoCOM chooses during start-up the default transmission data-format. If the TPS Firmware Release is higher than 2.20 then, this will be binary mode. Nevertheless, one can force ASCII data format for special purposes, e.g. debugging.

In the case of an ASCII request, the server side will be set only. The server always replies in the data-format that it has received the request. In this context this RPC can be used to deny binary data format.

Parameters

EComFormat	Out	COM_ASCII or COM_BINARY

Return Codes

RC_OK	On successful completion.
RC_COM_PORT_NOT_OPEN	Port not open for transmission.
RC_COM_NO_BINARY	TPS Firmware does not support binary data transmission format.

See Also

COM_SetComFormat

Example

```
RC_TYPE     rc;
COM_FORMAT  eFormat;

// change coding method
// eFormat is COM_ASCII or COM_BINARY
eFormat = COM_BINARY;
rc = COM_SetComFormat(eFormat);
if (rc == RC_COM_PORT_NOT_OPEN)
{
  rc = COM_SetComFormat(eFormat);
}

switch (rc)
{
  case RC_COM_PORT_NOT_OPEN:
    printf("Port not open\n");
    return (RC_FATAL);
    break;

  case RC_COM_NO_BINARY:
    printf("Binary format not available "
           "for this version.");
    // continue in ASII-format
    break;

}  // end of switch (rc)

// continue in program
```

5.3.11 COM_UseWindow - Declare Parent Window Handle

C-Declaration

```
COM_UseWindow( HWND handle )
```

VB-Declaration

```
VB_COM_UseWindow( handle As HWND )
```

ASCII-Request

-

ASCII-Response

-

Remarks

The function sets the parent window-handle that GeoCOM uses when it creates a dialog or message box. If this function is not called, GeoCOM will use the `NULL` window as default.

> **Note:** `HWND` depends on whether the pre-processor symbol `STRICT` is defined. When MFC libraries are used, `STRICT` is automatically defined. Otherwise the user must `#define STRICT` or he will get unresolved externals.

Parameters

`handle`	`In`	Parent window handle.

Return Codes

`RC_OK`	On successful completion.

See Also

-

Example

```
RC_TYE  rc;
HWND    hWnd;

rc = COM_UseWindow(hWnd);
```

5.3.12 COM_SetConnDlgFlag - Set Connection Dialog Flag

C-Declaration

```
COM_SetConnDlgFlag( BOOLE bShow )
```

VB-Declaration

```
VB_COM_SetConnDlgFlag( ByVal bShow As Integer )
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function sets or clears the connection dialog flag. If cleared, then the user will not be asked by a dialog box if all possible settings should be tried to establish a connection. The flag is set to TRUE by default. Usually this function will be called beforehand of `COM_OpenConnection`.

Parameters

`bShow`	`In`	show dialog

Return Codes

`RC_OK`	On successful completion.

See Also

`COM_OpenConnection`

Example

```
Const short  RETRIES = 2;
RC_TYPE      rc;

// disable dialog box
COM_SetConnDlgFlag(FALSE);
// establish connection without showing dialog box
rc = COM_OpenConnection(PORT_1, COM_BAUD_9600, Retries);
```

5.3.13 COM_ViewError - Pop Up Error Message Box

C-Declaration

```
COM_ViewError( RC_TYPE  Result,
               char    *szMsgTitle )
```

VB-Declaration

```
VB_COM_ViewError( ByVal Result      As Integer,
                  ByVal szMsgTitle  As String)
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function checks the value of Result and if it is not equal to `RC_OK` then it pops up a message box containing the specific error text.

Note: This function yields a valid error text only if GeoCOM has been initialised successfully.

Parameters

`Result`	`In`	Error result code.
`szMsgTitle`	`In`	Title of the displayed dialog box.

Return Codes

`RC_OK`	Always.

See Also

`COM_GetErrorText`

Example

```
RC_TYPE    rc;

// initialize GeoCOM
rc = COM_SetBaudRate(COM_BAUD_19200);

if (rc != RC_OK)
{
  COM_ViewError(rc, "Set up connection");
  // handle error
}
```

5.3.14 COM_GetErrorText - Get Error Text

C-Declaration

```
COM_GetErrorText( RC_TYPE Result,
```

```
                        char    *szErrText)
```

VB-Declaration

```
VB_COM_GetErrorText(ByVal Result     As Integer,
                          szErrText  As String)
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function checks the value of Result and returns an error text if the value is not equal to RC_OK. The function yields an empty string if the value is RC_OK. The maximum length of such an error text is 255 characters.

Parameters

`Result`	`In`	Error code of a function called before this code will be checked.
`szErrText`	`Out`	Error text if not equal to `RC_OK`.

Return Codes

RC_OK	On successful completion.

See Also

`COM_ViewError`

5.3.15 COM_GetWinSWVersion - Retrieve Client Version Information

C-Declaration

```
COM_GetWinSWVersion( short &nRel,
                     short &nVer,
                     short &nSubVer )
```

VB-Declaration

```
VB_COM_GetWinSWVersion( nRel    As Integer,
                        nVer    As Integer,
                        nSubVer As Integer )
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function retrieves the actual software Release (Release, version and subversion) of GeoCOM on the client side.

Parameters

nRel	Out	Software Release.
nVer	Out	Software version.
nSubVer	Out	Software subversion.

Return Codes

RC_OK	On successful completion.

See Also

COM_GetSWVersion

Example

```
RC_TYPE   rc;
short     nRel, nSubVer, nVer;

 (void) COM_GetWinSWVersion(nRel, nVer, nSubVer);

printf(„Windows GeoCOM:\n“);

printf(„Release %2d.%02d.%02d\n“, nRel, nVer, nSubVer);
```

5.3.16 COM_GetTPSState – Get Current TPS Operation Mode

C-Declaration

```
COM_GetTPSState( COM_TPS_STATUS &eMode )
```

VB-Declaration

```
VB_COM_ GetTPSState( eMode As Integer )
```

ASCII-Request

-

ASCII-Response

-

Remarks

This function retrieves the current operation mode.

Note: This function returns a valid value only if the sign-off message is enabled. In any other situation, it returns COM_TPS_UNKNOWN.
It is important to enable the sign-off message again if the instrument has been shut down.

Parameters

eMode	Out	Current operation mode.

Return Codes

RC_OK	On successful completion.

See Also

COM_EnableSignOff

CTL_GetUpCounter

Example

```
RC_TYPE          rc;
COM_TPS_STATUS   eMode;

rc = COM_EnableSignOff(TRUE);
if (rc == RC_OK)
{
  rc = COM_GetTPSSTate(eMode);
  if (rc == RC_OK)
  {
    switch (eMode)
    {
       case COM_TPS_OFF:
         printf("TPS is switched off\n");
         break;

       ...

       case COM_TPS_UNKNOWN:
       default:
         printf("TPS state unknown\n");
    }
  }
}
```

6 AUTOMATION - AUT

6.1 USAGE

The subsystem ‘Automation’ mainly performs the dynamic application ‘absolute positioning’. This operation positions the axes of the instrument within a given tolerance to the system’s angle measurement unit.

In combination with the Automatic Target Recognition System (ATR) other functionality such as automatic target position or target search are supported.

Some of the functions of this subsystem can take a undefined time for execution (for example the position operation takes the more time the more precision is required).

6.2 CONSTANTS AND TYPES

On/Off switch

```
enum ON_OFF_TYPE
{
  OFF,                    // 0
  ON                      // 1
};
```

Number of axis

```
const short MOT_AXES = 2;
```

Positioning Tolerance

```
struct AUT_POSTOL
{
  double  adPosTol[MOT_AXES];
      // positioning tolerance for Hz and V [rad]
};
```

Maximum Position Time [s]

```
struct AUT_TIMEOUT
{
 double adPosTimeout[MOT_AXES];// max. positioning time [sec]
};
```

Position Precision

```
enum AUT_POSMODE
{
        AUT_NORMAL = 0,     // fast positioning mode
        AUT_PRECISE = 1     // exact positioning mode
                            // note: can distinctly claim more time
                            //       for the positioning
}
```

Fine-adjust Position Mode

```
enum AUT_ADJMODE            // Possible settings of the positioning
                            // tolerance relating the angle- or the
                            // point- accuracy at the fine adjust.
{
        AUT_NORM_MODE = 0   // Angle tolerance
        AUT_POINT_MODE = 1  // Point tolerance
}
```

Automatic Target Recognition Mode

```
enum AUT_ATRMODE            // Possible modes of the target
                            // recognition
{
        AUT_POSITION = 0,   // Positioning to the hz- and v-angle
        AUT_TARGET = 1      // Positioning to a target in the
                            // environment of the hz- and v-angle.
}
```

6.3 FUNCTIONS

6.3.1 AUT_GetATRStatus - Get the status of the ATR mode

C-Declaration

```
AUT_GetATRStatus(ON_OFF_TYPE &OnOff)
```

VB-Declaration

```
VB_AUT_GetATRStatus(OnOff As Integer)
```

ASCII-Request

```
%R1Q,9019:
```

ASCII-Response

`%R1P,0,0:`*RC,OnOff[long]*

Remarks

Get the current status of the ATR mode on TCA instruments. This command does not indicate whether the ATR has currently acquired a prism.

Parameters

`OnOff`	`out`	Status of the ATR mode

Return-Codes

`RC_OK`	Execution always successful.

See Also

```
AUT_SetATRStatus
```

Example

```
RC_TYPE         rc;
ON_OFF_TYPE     OnOff;

// look for ATR status and set On if it is Off

rc = AUT_GetATRStatus(OnOff);
if (OnOff == OFF)
{
  rc = AUT_SetATRStatus(ON);
  if (rc == RC_OK)
  {
    // set of ATR status successful
  }
  else
```

```
  {
    // no TCA instrument
  }
}
```

6.3.2 AUT_SetATRStatus - Set the status of the ATR mode

C-Declaration

`AUT_SetATRStatus(ON_OFF_TYPE OnOff)`

VB-Declaration

`VB_AUT_SetATRStatus(OnOff As Integer)`

ASCII-Request

`%R1Q,9018:`*OnOff[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

Activate respectively deactivate the ATR mode.

Activate ATR mode:
The ATR mode is activated and the LOCK mode keep unchanged.

Deactivate ATR mode:
The ATR mode is deactivated and the LOCK mode (if sets) will be reset automatically also.

This command is valid for TCA instruments only.

Parameters

`OnOff`	`in`	Status of the ATR mode

Return-Codes

`RC_OK`	Execution successful.
`RC_NOT_IMPL`	ATR not available; no TCA instrument.
`AUT_RC_DETECTOR_ERROR`	Indefinite ATR error

See Also

`AUT_GetATRStatus`
`AUT_GetLockStatus`
`AUT_SetLockStatus`

Example

`see AUT_GetATRStatus`

6.3.3 AUT_GetLockStatus - Get the status of the lock switch

C-Declaration

```
AUT_GetLockStatus(ON_OFF_TYPE &OnOff)
```

VB-Declaration

```
VB_AUT_GetLockStatus(OnOff As Integer)
```

ASCII-Request

```
%R1Q,9021:
```

ASCII-Response

`%R1P,0,0:`*RC, OnOff[long]*

Remarks

This command gets the current LOCK switch. This command is valid for TCA instruments only and does not indicate whether the ATR has a prism in lock or not.
With the function `MOT_ReadLockStatus` you can find out whether a target is locked or not.

Parameters

`OnOff`	`Out`	Status of the ATR lock switch

Return-Codes

`RC_OK`	Execution always successful.

See Also

```
AUT_SetLockStatus
MOT_ReadLockStatus
```

Example

```
RC_TYPE        rc;
ON_OFF_TYPE    OnOff, OldAtrStatus;

rc = AUT_GetATRStatus(OldAtrStatus); // save old mode
rc = AUT_GetLockStatus(OnOff);

if (OnOff == OFF)
{// ------ enable target tracking -----------------
  rc = AUT_SetLockStatus(ON); //set the ATR mode
                              //automatically also!
  if (rc == RC_OK)
  {// set of Lock status successful
    rc = AUT_LockIn();// activate the real target
                      // tracking
```

```
        if(rc != RC_OK)
        {
           // error handling
        }
      }
      else
      {
        // no TCA instrument
      }
    }
    else
    {// ------ disable target tracking ---------------
        rc = AUT_SetLockStatus(OFF);// reset the ATR
                                    // mode not
                                    // automatically
        if(rc == RC_OK)
        {// reset of Lock status successful
           if(OldAtrStatus==OFF)
           {// set old ATR mode
           rc == AUT_SetATRStatus(OFF);
           }
        }
    }
```

6.3.4 AUT_SetLockStatus - Set of the ATR lock switch

C-Declaration

`AUT_SetLockStatus(ON_OFF_TYPE OnOff)`

VB-Declaration

`VB_AUT_SetLockStatus(OnOff As Integer)`

ASCII-Request

`%R1Q,9020:`*OnOff[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

Set the lock status.

Status ON:
The target tracking functionality is available but not activated. In order to activate target tracking, see the function `AUT_LockIn`. The ATR mode will be set automatically.

Status OFF:
A running target tracking will be aborted and the manual driving wheel is activated. The ATR mode will be not reset automatically respectively keep unchanged.

This command is valid for TCA instruments only.

Parameters

`OnOff`	`in`	Status of the ATR lock switch

Return-Codes

`RC_OK`	Execution successful.
`RC_NOT_IMPL`	AZE not available; no TCA instrument.
`AUT_RC_DETECTOR_ERROR`	Indefinite ATR error

See Also

`AUT_GetLockStatus`
`AUT_SetATRStatus`
`AUT_LockIn`

Example

`see AUT_GetLockStatus`

6.3.5 AUT_ReadTol - Read current setting for the positioning tolerances

C-Declaration

```
AUT_ReadTol(AUT_POSTOL &TolPar)
```

VB-Declaration

```
VB_AUT_ReadTol(TolPar As AUT_POSTOL)
```

ASCII-Request

`%R1Q,9008:`

ASCII-Response

`%R1P,0,0:`*RC,Tolerance Hz[double],Tolerance V[double]*

Remarks

This command reads the current setting for the positioning tolerances of the Hz- and V- instrument axis.

This command is valid for TCM and TCA instruments only.

Parameters

TolPar	out	The values for the positioning tolerances in Hz and V direction [rad].

Return-Codes

RC_OK	Execution always successful.

See Also

AUT_SetTol

Example

```
const double MIN_TOL=3.141592654e-05;

RC_TYPE         rc;
AUT_POSTOL      TolPar;

// read tolerance and set to a minimum of
// 3.141592654e-05

rc = AUT_ReadTol(TolPar);

if ((TolPar.adPosTol[MOT_HZ_AXLE] > MIN_TOL) ||
   (TolPar.adPosTol[MOT_V_AXLE] > MIN_TOL))
{
  TolPar.adPosTol[MOT_HZ_AXLE] = MIN_TOL;
  TolPar.adPosTol[MOT_HZ_AXLE] = MIN_TOL;
  rc = AUT_SetTol(TolPar);
  switch (rc)
  {
    case (RC_OK):
      // set of Lock tolerance successful
      break;
    case (RC_IVPARAM):
      // invalid parameter
      break;
    case (MOT_RC_UNREADY):
      // subsystem not ready
      break;
  }
}
```

6.3.6 AUT_SetTol - Set the positioning tolerances

C-Declaration

```
AUT_SetTol(AUT_POSTOL  TolPar)
```

VB-Declaration

```
VB_AUT_SetTol(TolPar As AUT_POSTOL)
```

ASCII-Request

`%R1Q,9007:`*ToleranceHz[double], Tolerance V[double]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This command stops every movement and sets new values for the positioning tolerances of the Hz- and V- instrument axes. This command is valid for TCM and TCA instruments only.

The tolerances must be in the range of 1[cc] (=1.57079 E-06[rad]) to 100[cc] (=1.57079 E-04[rad]).

Note: The max. Resolution of the angle measurement system is 3.2[cc] for the TPS1000 instruments and 1.6[cc] for the TPS2000 instruments.
If smaller positioning tolerances are required, the positioning time can increase drastically.

Parameters

TolPar	in	The values for the positioning tolerances in Hz and V direction [rad].

Return-Codes

RC_OK	Execution successful.
RC_IVPARAM	One or both tolerance values not within the boundaries (1.57079E-06[rad] =1[cc] to 1.57079E-04[rad] =100[cc]).
MOT_RC_UNREADY	Instrument has no motorization

See Also

AUT_ReadTol

Example

see AUT_ReadTol

6.3.7 AUT_ReadTimeout - Read current timeout setting for positioning

C-Declaration

AUT_ReadTimeout(AUT_TIMEOUT &TimeoutPar)

VB-Declaration

```
VB_AUT_ReadTimeout(TimeoutPar As AUT_TIMEOUT)
```

ASCII-Request

```
%R1Q,9012:
```

ASCII-Response

`%R1P,0,0:`*RC, TimeoutHz[double], TimeoutV[double]*

Remarks

This command reads the current setting for the positioning time out (maximum time to perform positioning).

Parameters

`TimeoutPar`	`Out`	The values for the positioning time out in Hz and V direction [sec].

Return-Codes

`RC_OK`	Execution always successful.

See Also

`AUT_SetTimeout`

Example

```
RC_TYPE         rc;
AUT_TIMEOUT     TimeoutPar;

// read timeout and set to a minimum of 10 [s]

rc = AUT_ReadTimeout(TimeoutPar);

if ((TimeoutPar.adPosTimeout[0] < 10) ||
   (TimeoutPar.adPosTimeout[1] < 10))
{
  TimeoutPar.adPosTimeout[0] = 10;
  TimeoutPar.adPosTimeout[1] = 10;
  rc = AUT_SetTimeout(TimeoutPar);
  switch (rc)
  {
    case (RC_OK):
      // set of timeout successful
```

```
        break;
      case (RC_IVPARAM):
        // invalid parameter
        break;
    }
  }
```

6.3.8 AUT_SetTimeout - Set timeout for positioning

C-Declaration

`AUT_SetTimeout(AUT_TIMEOUT TimeoutPar)`

VB-Declaration

`VB_AUT_SetTimeout(TimeoutPar As AUT_TIMEOUT)`

ASCII-Request

`%R1Q,9011:`*TimeoutHz[double],TimeoutV[double]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This command set the positioning timeout (set maximum time to perform a positioning). The timeout is reset on 10[sec] after each power on

Parameters

`TimeoutPar`	`in`	The values for the positioning timeout in Hz and V direction [s]. Valid values are between 1 [sec] and 60 [sec].

Return-Codes

`RC_OK`	Execution successful. Maximum positioning time is set.
`RC_IVPARAM`	One or both time out values not within the boundaries (1[sec] to 60[sec]).

See Also

`AUT_ReadTimeout`

Example

see `AUT_ReadTimeout`

6.3.9 AUT_MakePositioning - Turns telescope to specified position

C-Declaration

```
AUT_MakePositioning(double Hz,
                    double V,
                    AUT_POSMODE POSMode,
                    AUT_ATRMODE ATRMode,
                    BOOLE bDummy)
```

VB-Declaration

```
VB_AUT_MakePositioning4(Hz As Double,
                        V As Double,
                        POSMode As Integer,
                        ATRMode As Integer,
                        bDummy As Boolean)
```

ASCII-Request

%R1Q,9027:*Hz,V,PosMode,ATRMode,0*

ASCII-Response

%R1P,0,0:*RC*

Remarks

This procedure turns the telescope absolute to the in *Hz* and *V* specified position, taking tolerance settings for positioning (see AUT_POSTOL) into account. Any active control function is terminated by this function call.

If the position mode is set to normal (PosMode = AUT_NORMAL) it is assumed that the current value of the compensator measurement is valid. Positioning precise (PosMode = AUT_PRECISE) forces a new compensator measurement at the specified position and includes this information for positioning.

If ATR is possible and activated and the ATR mode is set to AUT_TARGET, the instrument tries to position onto a target in the destination area. In addition, the target is locked after positioning if the LockIn status is set. If the Lock status not set, the manual driving wheel is activated after the positioning.

Parameters

Hz	In	Horizontal (telescope) position [rad].
V	In	Vertical (instrument) position [rad].

POSMode	In	Position mode: AUT_NORMAL: (default) uses the current value of the compensator (no compensator measurement while positioning). For values >25GON positioning might tend to inaccuracy. AUT_PRECISE: tries to measure exact inclination of target. Tend to longer position time (check AUT_TIMEOUT and/or COM-time out if necessary).
ATRMode	In	Mode of ATR: AUT_POSITION: (default) conventional position using values Hz and V. AUT_TARGET: tries to position onto a target in the destination area. This mode is only possible if ATR exists and is activated.
bDummy	In	It's reserved for future use, set bDummy always to FALSE

Return-Codes

RC_OK	Execution successful.
RC_IVPARAM	Invalid parameter (e.g. no valid position).
AUT_RC_DETENT_ERROR	Destination angle lies in the illegal sector, collision is occurred. Set new Destination angle or free illegal sectors.
AUT_RC_TIMEOUT	Time out while positioning of one or both axes. (perhaps increase AUT time out, see AUT_SetTimeout)
AUT_RC_MOTOR_ERROR	Instrument has no 'motorization'.
AUT_RC_ANGLE_ERROR	Error within angle measurement. Maybe instrument is not levelled and not stable. Try to correct instrument's position or switch automatic incline correction to OFF (see TMC_SetInclineSwitch).

AUT_RC_INCACC	In the position mode AUT_PRECISE a valid measurement of target's inclination was not possible: position inexact.
RC_ABORT	Function aborted.
RC_COM_TIMEDOUT	Communication timeout. (perhaps increase COM timeout, see COM_SetTimeout)

Additionally with position mode AUT_TARGET.

AUT_RC_NO_TARGET	No target found
AUT_RC_MULTIPLE_TARGETS	Multiple targets found.
AUT_RC_BAD_ENVIRONMENT	Inadequate environment conditions.
AUT_RC_ACCURACY	Inexact fine position, repeat positioning
AUT_RC_DEV_ERROR	During the determination of the angle deviation error detected, repeat positioning
AUT_RC_NOT_ENABLED	ATR mode not enabled, enable ATR mode

See Also

AUT_GetATRStatus, AUT_SetATRStatus
AUT_GetLockStatus, AUT_SetLockStatus
AUT_ReadTol, AUT_SetTol
AUT_ReadTimeout, AUT_SetTimeout
COM_GetTimeOut, COM_SetTimeOut

Example

The example program tries to positioning to the given position. If a time out occurred, the time out values are increased and the position procedure starts again. If a measurement error occurred, the automatic inclination correction is switched off and the position procedure starts again.

```
RC_TYPE         rc, hrc;
short           i;
BOOL            TryAgain = TRUE;
AUT_TIMEOUT     TimeoutPar;
AUT_POSMODE     POSMode = AUT_PRECISE;
short           nComTimeOut, nOldComTimeOut;
```

```
rc=RC_IVRESULT;
hrc = COM_GetTimeOut(nOldComTimeOut);
hrc =AUT_SetATRStatus(ON); // for the ATR mode
                           // AUT_TARGET necessary,
                           // otherwise not necessary

while(rc!=RC_OK || TryAgain)
{
  rc = AUT_MakePositioning(1.3, 1.6, POSMode,
                           AUT_TARGET, FALSE );
  switch (rc)
  {
    case (RC_OK):
      //Positioning successful and precise
      break;
    case (AUT_RC_TIMEOUT):
      // measure timeout fault: increase timeout
      hrc = AUT_ReadTimeout(TimeoutPar);
      TimeoutPar.adPosTimeout[0]
        = __min(TimeoutPar.adPosTimeout[0]+=5,60);
      TimeoutPar.adPosTimeout[1]
        = __min(TimeoutPar.adPosTimeout[1]+=5,60);
      hrc = AUT_SetTimeout(TimeoutPar);
      break;
    case RC_COM_TIMEDOUT:
      //increase timeout
      hrc = COM_GetTimeOut(nComTimeOut);
      nComTimeOut=__min(nComTimeOut+=5, 60);
      hrc = COM_SetTimeOut(nComTimeOut);
      break;
    case AUT_RC_ANGLE_ERROR:
      // error within angle measurement:
      // switch inclination correction off
      hrc = TMC_SetInclineSwitch(OFF);
      break;
    default:
      // precise position not possible
      TryAgain = FALSE;
      if (rc == AUT_RC_INCACC)
      {
        //Position successful but not precise
      }
      else
      {
```

```
          // Positioning not successful
          // here further error analyse possible
        }
        break;
    }
}

rc = AUT_SetATRStatus(OFF); // Note: LOCK mode will
                            // be automatically
                            // reseted !
hrc = COM_SetTimeOut(nOldComTimeOut);// Set old time-
                                     // out
```

6.3.10 AUT_ChangeFace - Turns telescope to other face

C-Declaration

```
AUT_ChangeFace(AUT_POSMODE PosMode,
               AUT_ATRMODE ATRMode,
               BOOLE bDummy)
```

VB-Declaration

```
VB_AUT_ChangeFace4(PosMode As Integer,
                   ATRMode As Integer,
                   bDummy As Boolean)
```

ASCII-Request

`%R1Q,9028:`*PosMode,ATRMode,0*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This procedure turns the telescope to the other face.
Is in the moment of the function calling an other control function active it will be terminated before.

The start angle is automatically measured before the position starts.

If the position mode is set to normal (`PosMode = AUT_NORMAL`) it is allowed that the current value of the compensator measurement is inexact. Positioning precise (`PosMode = AUT_PRECISE`) forces a new compensator measurement. If this measurement is not possible, the position does not take place.

If ATR is possible and activated and the ATR mode is set to AUT_TARGET the instrument tries to position onto a target in the destination area. In addition, the target is locked after positioning if the LockIn status is set.

Parameters

POSMode	In	Position mode: AUT_NORMAL: uses the current value of the compensator. For values >25GON positioning might tend to inexact. AUT_PRECISE: tries to measure exact inclination of target. Tends to long position time (check AUT_TIMEOUT and/or COM-time out if necessary).
ATRMode	In	Mode of ATR: AUT_POSITION: conventional position to other face. AUT_TARGET: tries to position onto a target in the destination area. This set is only possible if ATR exists and is activated.
bDummy	In	It's reserved for future use, set bDummy always to FALSE

Return-Codes

General:

RC_OK	Execution successful.
RC_IVPARAM	Invalid parameter.
AUT_RC_DETENT_ERROR	Destination angle lies in the illegal sector, collision is occurred. Set new Destination angle.
AUT_RC_TIMEOUT	Timeout while positioning of one or both axes. (perhaps increase AUT timeout, see AUT_SetTimeout)
AUT_RC_MOTOR_ERROR	Instrument has no 'motorization'.

AUT_RC_ANGLE_ERROR	Error within angle measurement. Maybe instrument is not levelled and not stable. Try to correct instrument's position or switch automatic incline correction to OFF (see TMC_SetInclineSwitch).
AUT_RC_INCACC	In the position mode AUT_PRECISE a valid measurement of target's inclination was not possible. The positioning is inexact.
RC_FATAL	Fatal error.
RC_ABORT	Function aborted.
RC_COM_TIMEDOUT	Communication timeout. (perhaps increase COM timeout, see COM_SetTimeout)

Additionally with position mode AUT_TARGET.

AUT_RC_NO_TARGET	No target found
AUT_RC_MULTIPLE_TARGETS	Multiple targets found.
AUT_RC_BAD_ENVIRONMENT	Inadequate environment conditions.
AUT_RC_ACCURACY	Inexact fine position, repeat positioning
AUT_RC_DEV_ERROR	During the determination of the angle deviation error detected, repeat change face
AUT_RC_NOT_ENABLED	ATR mode not enabled, enable ATR mode

See Also

AUT_GetATRStatus, AUT_SetATRStatus
AUT_GetLockStatus, AUT_SetLockStatus
AUT_ReadTol, AUT_SetTol
AUT_ReadTimeout, AUT_SetTimeout
COM_GetTimeOut, COM_SetTimeOut
TMC_GetFace

Example

The example program performs a change face. If a measurement error occurs, the automatic inclination correction is switched off and the change face starts again.

```
RC_TYPE         rc, rch;
BOOL            TryAgain = TRUE;
AUT_POSMODE     POSMode = AUT_PRECISE;

rc=RC_IVRESULT;

while(rc!=RC_OK && TryAgain)
{
  rc = AUT_ChangeFace(POSMode,
                      AUT_POSITION,
                      FALSE);
  switch (rc)
  {
  case (RC_OK): // position successful
    //change face successful and precise
    break;
  case (AUT_RC_ANGLE_ERROR):
    //error within angle measurement:
    //switch inclination correction off
    rch = TMC_SetInclineSwitch(OFF);
    break;
  case (RC_COM_TIMEDOUT):
    //communication timed out while change face
    TryAgain = FALSE;
    break;
  default:
    //precise position not possible
    TryAgain = FALSE;
    if (rc == AUT_RC_INCACC)
    {
      //change face successful but not precise
    }
    else
    {
      // change face not successful
      // here further error analyse possible
    }
    break;
  }
}
```

6.3.11 AUT_FineAdjust - Automatic target positioning

C-Declaration

```
AUT_FineAdjust(       Double dSrchHz,
                      double dSrchV ,
                      BOOLE  bDummy)
```

VB-Declaration

```
VB_AUT_FineAdjust3(       DSrchHz As Double,
                          dSrchV  As Double,
                          bDummy As Boolean)
```

ASCII-Request

`%R1Q,9037:` `dSrchHz`*[double],* `dSrchV`*[double],0*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This procedure performs a positioning of the Theodolite axis onto a destination target. If the target is not within the sensor measure region a target search will be executed. The target search range is limited by the parameter `dSrchV` in V- direction and by parameter `dSrchHz` in Hz - direction. If no target found the instrument turns back to the initial start position. The ATR mode must be enabled for this functionality, see `AUT_SetATRStatus` and `AUT_GetATRStatus`.

Any actual target lock is terminated by this procedure call. After position, the target is not locked again.

The timeout of this operation is set to 5s, regardless of the general position timeout settings. The positioning tolerance is depends on the previously set up the fine adjust mode (see `AUT_SetFineAdjustMoed` and `AUT_GetFineAdjustMode`).

Tolerance settings (with `AUT_SetTol` and `AUT_ReadTol`) have no influence to this operation. The tolerance settings as well as the ATR measure precision are depends on the instrument's class and the used EDM measure mode (The EDM measure modes are handled by the subsystem TMC):

EDM measure mode	**Instr. Class, measure precision / tolerance**			
	1100	**1700**	**1800**	**5000**
EDM_SINGLE_FAST	35/40[cc]	25/40[cc]	21/40[cc]	18/36[cc]
EDM_SINGLE_STANDARD	10/20[cc]	5/10[cc]	3/6[cc]	2/5[cc]
EDM_SINGLE_EXACT	5/10[cc]	2.5/5[cc]	2/5[cc]	2/5[cc]
EDM_CONT_FAST	40/40[cc]	40/40[cc]	40/40[cc]	40/40[cc]
EDM_CONT_STANDARD	40/40[cc]	40/40[cc]	36/40[cc]	33/40[cc]
EDM_CONT_EXACT	5/10[cc]	2.5/5[cc]	2/5[cc]	2/5[cc]

Parameters

DSrchHz	In	Search range Hz-
DSrchV	In	Search range V-axis
bDummy	In	It's reserved for future use, set bDummy always to FALSE

Return-Codes

RC_OK	Execution successful.
AUT_RC_TIMEOUT	Timeout while positioning of one or both axes. The position fault lies above 100[cc]. (perhaps increase AUT timeout, see AUT_SetTimeout)
AUT_RC_MOTOR_ERROR	Instrument has no 'motorization'.
RC_FATAL	Fatal error.
RC_ABORT	Function aborted.
AUT_RC_NO_TARGET	No target found.
AUT_RC_MULTIPLE_TARGETS	Multiple targets found.
AUT_RC_BAD_ENVIRONMENT	Inadequate environment conditions.
AUT_RC_DEV_ERROR	During the determination of the angle deviation error detected, repeat fine positioning
AUT_RC_NOT_ENABLED	ATR mode not enabled, enable ATR mode
AUT_RC_DETECTOR_ERROR	AZE error, at repeated occur call service

RC_COM_TIMEDOUT	Communication time out. (perhaps increase COM timeout, see COM_SetTimeout)

See Also

AUT_SetATRStatus
AUT_GetATRStatus
AUT_SetFineAdjustMode
AUT_GetFineAdjustMode

Example

```
RC_TYPE     Result;
ON_OFF_TYPE ATRState;
double  dHzSearchRange, dVSearchRange

dHzSearchRange=0.08;// search range in [rad]
dVSearchRange=0.08; // search range in [rad]

Result = AUT_GetATRStatus(ATRState);// The ATR-
                                    // Status must
                                    // be set for
                                    // the fine
                                    // adjust
                                    // functionality
if(ATRState==ON)
{
     // performs a fine position with a max. target
     // search range of 0.08rad (5gon) in Hz and V
     // direction
     Result = AUT_FineAdjust(dHzSearchRange,
                                    dVSearchRange,
                                    FALSE);
     switch (Result) // function return code
     {
       case (RC_OK):
         //fine adjust successful and precise
         break;
       case (AUT_RC_NO_TARGET):
         //no target found.
         break;
       case (AUT_RC_MULTIPLE_TARGETS):
         //multiple targets found.
         break;
       case (AUT_RC_BAD_ENVIRONMENT):
         //inadequate environment conditions.
         break;
```

```
        default:
          //fine adjust not successful
          //here further error analyse possible
          break;
      }
}
```

6.3.12 AUT_Search - Performs an automatically target search

C-Declaration

```
AUT_Search(double Hz_Area,
           double V_Area,
           BOOLE  bDummy)
```

VB-Declaration

```
VB_AUT_Search2(Hz_Area As Double,
               V_Area As Double,
               bDummy As Boolean)
```

ASCII-Request

`%R1Q,9029:`*Hz_Area,V_Area,0*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This procedure performs an automatically target search within a given area. The search area has an elliptical shape where the input parameters determine the axis in horizontal and vertical direction. If the search was successful the telescope will position to the target in a exactness of the field of vision (±1.25[GON]), otherwise the instrument turns back to the initial start position. With the ESC key a running search process will be aborted. The ATR mode must be enabled for this functionality, see `AUT_SetATRStatus()` and `AUT_GetATRStatus`. For a exact positioning use fine adjust (see `AUT_FineAdjust`) afterwards.

Note: If you expand the search range of the function `AUT_FineAdjust`, then you have a target search and a fine positioning in one function.

Parameters

`Hz_Area`	`In`	Horizontal search region [`rad`].
`V_Area`	`In`	Vertical search region [`rad`].

bDummy	In	It's reserved for future use, set bDummy always to FALSE

Return-Codes

RC_OK	Execution successful.
RC_IVPARAM	Invalid parameter.
AUT_RC_MOTOR_ERROR	Instrument has no 'motorization'.
RC_FATAL	Fatal error.
RC_ABORT	Function aborted.
AUT_RC_NO_TARGET	No target found.
AUT_RC_BAD_ENVIRONMENT	Inadequate environment conditions.
AUT_RC_NOT_ENABLED	ATR mode not enabled, enable ATR mode
AUT_RC_DETECTOR_ERROR	AZE error, at repeated occur call service
RC_COM_TIMEDOUT	Communication timeout. (perhaps increase COM timeout, see COM_SetTimeout)

See Also

AUT_SetATRStatus
AUT_GetATRStatus
AUT_FineAdjust

Example

The example program performs a search in the given area. If no target is found, the area is increased until 1[rad]. If a communication timeout occurs, the value for the communication timeout is increased until 30 [s] (Note that a search over a big area takes a long time often results in an error).

```
RC_TYPE    rc, hrc;
BOOL       TryAgain = TRUE;
double     Hz_Area,V_Area;
short      nComTimeOut, nOldComTimeOut;

Hz_Area = 0.1;
V_Area  = 0.1;
rc = RC_IVRESULT;

hrc = COM_GetTimeOut(nOldComTimeOut);
hrc =AUT_SetATRStatus(ON);  // activate ATR mode
```

```
while(rc!=RC_OK && TryAgain && hrc==RC_OK)
{
  rc = AUT_Search(Hz_Area,V_Area,FALSE);
  switch (rc)
  {
  case (RC_OK):
    // execution successful
    // Target found
    break;
  case (AUT_RC_NO_TARGET):
    //no target found.
    //increase search area
    Hz_Area += 0.1;
    V_Area += 0.1;
    if (Hz_Area > 1)
    {
      TryAgain = FALSE;
    }
    break;
  case (RC_COM_TIMEDOUT):
    //communication timeout
    //increase timeout until 30s
    hrc = COM_GetTimeOut(nComTimeOut);
    nComTimeOut=(short)__min(nComTimeOut+=5, 60);
    hrc = COM_SetTimeOut(nComTimeOut);
    //abort if timeout >= 30s
    if (nComTimeOut >= 30)
    {
      TryAgain = FALSE;
    }
    break;
  default:
    //error: search not possible
    //here further error analyse possible
    break;
  }
}

hrc = COM_GetTimeOut(nOldComTimeOut);// Set old time
                                     // out back
hrc = AUT_SetATRStatus(OFF);// Note: LOCK mode will
                            // be automatically also
                            // reseted!
```

6.3.13 AUT_GetFineAdjustMode - Get fine adjust positioning mode

C-Declaration

```
AUT_GetFineAdjustMode(AUT_ADJMODE& rAdjMode)
```

VB-Declaration

```
VB_AUT_GetFineAdjustMode(AdjMode As Integer)
```

ASCII-Request

```
%R1Q,9030:
```

ASCII-Response

```
%R1P,0,0:RC,AdjMode[integer]
```

Remarks

This function returns the current activated fine adjust positioning mode. This command is valid for all instruments, but has only effects for TCA instruments.

Parameters

`RAdjMode`	Out	current fine adjust positioning mode

Return-Codes

`RC_OK`	Execution successful (always)

See Also

```
AUT_SetFineAdjustMode
```

Example

```
see AUT_SetFineAdjustMode
```

6.3.14 AUT_SetFineAdjustMode - Set the fine adjustment mode

C-Declaration

```
AUT_SetFineAdjustMode(AUT_ADJMODE AdjMode)
```

VB-Declaration

```
VB_AUT_SetFineAdjustMode(AdjMode As Integer)
```

ASCII-Request

```
%R1Q,9031:AdjMode[long]
```

ASCII-Response

```
%R1P,0,0:RC
```

Remarks

This function sets the positioning tolerances (default values for both modes) relating the angle accuracy or the point accuracy for the fine adjust. This command is valid for all instruments, but has only effects for TCA instruments. It's to recommend to sets the adjust-mode to AUT_POINT_MODE, while a target is very near or held by hand.

Parameters

AdjMode	in	AUT_NORM_MODE:	Fine positioning with angle tolerance
		AUT_POINT_MODE;	Fine positioning with point tolerance

Return-Codes

RC_OK	Execution successful
RC_IVPARAM	Invalid mode

See Also

AUT_GetATRStatus

Example

```
RC_TYPE         Result;
AUT_ADJMODE     AdjMode;

Result=AUT_GetFineAdjustMode(AdjMode);
if(AdjMode!=AUT_MODE_POINT && Result==RC_OK)
{// change the finepositioning mode to AUT_MODE_POINT
     Result=AUT_SetFineAdjustMode(AUT_MODE_POINT);
     if(Result!=RC_OK)
     { // Error handling
     }
}
```

6.3.15 AUT_LockIn - Starts the target tracking

C-Declaration

AUT_LockIn()

VB-Declaration

VB_AUT_LockIn()

ASCII-Request

%R1Q,9013:

ASCII-Response

%R1P,0,0:*RC*

Remarks

Function starts the target tracking. Is at this time another ATR-configuration active, this configuration will be aborted before. The function can be called several times respectively is the target already locked, the command will be ignored. The LOCK mode must be enabled for this functionality, see AUT_SetLockStatus and AUT_GetLockStatus. So the ATR can lock the target, the target has to be in the field of view (FoV).

Parameters

-

Return-Codes

RC_OK	LOCK-IN configuration is now active or already active.
AUT_RC_NOT_ENABLED	Target acquisition not enabled
AUT_RC_MOTOR_ERROR	Instrument has no 'motorization'.
AUT_RC_DETECTOR_ERROR	Error in target acquisition, at repeated occur call service
AUT_RC_NO_TARGET	No target detected
AUT_RC_BAD_ENVIRONMENT	Bad environment conditions

See Also

AUT_SetLockStatus
AUT_GetLockStatus
MOT_ReadLockStatus

Example

```
RC_TYPE   result;

result = AUT_SetLockStatus(ON);// enable lock mode
if(result==RC_OK)
{
     result = AUT_LockIn(); // activate target tracking
     if(result != RC_OK)
     {
          // Error handling
     }
}
```

7 BASIC APPLICATIONS - BAP

The subsystem basic applications (BAP) contain high level functions for application programs.

7.1 CONSTANTS AND TYPES

Measurement Program Modes

```
enum BAP_MEASURE_PRG
  {
  BAP_NO_MEAS   = 0 // no measurements, take last one
  BAP_NO_DIST   = 1 // no dist. measurement, angles only
  BAP_DEF_DIST  = 2 // default distance measurements
  BAP_TRK_DIST  = 3 // tracking distance measurements,
                    // program and angles
  BAP_RTRK_DIST = 4 // rapid tracking distance
                    // measurements,
                    // program and angles *)
  BAP_CLEAR_DIST= 5 // clear distances
  BAP_STOP_TRK  = 6 // stop tracking
  };
```

*) Usage in ASCII mode depends on hardware

7.2 FUNCTIONS

7.2.1 BAP_GetLastDisplayedError - Get last TPS system error number

C-Declaration

```
BAP_GetLastDisplayedError(short &nError,
                          short &nGSIError)
```

VB-Declaration

```
VB_BAP_GetLastDisplayedError(nError As Integer,
                             nGSIError As Integer)
```

ASCII-Request

`%R1Q,17003:`

ASCII-Response

`%R1P,0,0:`*RC, nError[short], nGSIError[short]*

Remarks

TPS Firmware versions grater than 2.11 reset the last displayed error. So a second `GetLastDisplayedError` call will result in RC_OK.

Parameters

`nError`	out	Last displayed error-, info- or warning-number
`nGSIError`	out	Corresponding GSI error number

Return Codes

`RC_OK`	An error has been displayed before last call
`RC_IVRESULT`	No error has been displayed before last call

See Also

-

Example

```
RC_TYPE  rc;
short    nError, nGSIError;

rc = BAP_GetLastDisplayedError(nError, nGSIError);
if (rc == RC_OK)
  {
  printf("Error number: %d\n", nError);
  printf("GSI error number: %d\n", nGSIError);
  }
else if (rc == RC_IVRESULT)
  {
  printf("No error displayed before last call!");
  }
```

7.2.2 BAP_MeasDistanceAngle - Measure distance and angles

C-Declaration

```
BAP_MeasDistanceAngle(BAP_MEASURE_PRG &DistMode,
                      double &dHz, double &dV,
                      double &dDist)
```

VB-Declaration

```
VB_BAP_MeasDistAng(DistMode As Integer,
                   dHz As Double, dV As Double
                   dDist As Double)
```

ASCII-Request

`%R1Q,17017:`*DistMode[long]*

ASCII-Response

`%R1P,0,0:`*RC, dHz[double], dV[double], dDist[double],DistMode[long]*

Remarks

This function measures distances and angles depending on the mode `DistMode` and updates the internal datapool after correct measurements. It controls the special beep (sector or lost lock), maintains measurement icons and disables the "aF.."-key during tracking.

Parameters

`DistMode`	in	See Constants and Types
`DistMode`	out	BAP_DEF_DIST: unchanged
	out	BAP_TRK_DIST: unchanged
	out	BAP_RTRK_DIST: unchanged
	out	other modes: BAP_DEF_DIST
`dHz`	out	Horizontal angle [rad]x, depends on `DistMode`
`dV`	out	Vertical angle [rad]x, depends on `DistMode`
`dDist`	out	Slopedistance [m]x, depends on `DistMode`

Return Codes

`BAP_MeasDistanceAngle` may additionally return AUT- and TMC-return codes.

For detailed description of these return codes see manuals "Automatisation" (AUT) and "Theodolite Measurement and Calculation" (TMC)

RC_OK	Measurement executed successfully
AUT_RC_ANGLE_ERROR	Angle measurement error
AUT_RC_BAD_ENVIRONMENT	Bad Environment conditions
AUT_RC_CALACC	ATR-calibration failed
AUT_RC_DETECTOR_ERROR	Error in target acquisition
AUT_RC_DETENT_ERROR	Positioning not possible due to mounted EDM
AUT_RC_DEV_ERROR	Deviation measurement error
AUT_RC_INCACC	Position not exactly reached
AUT_RC_MOTOR_ERROR	Motorization error
AUT_RC_MULTIPLE_TARGETS	Multiple targets detected
AUT_RC_NO_TARGET	No target detected
AUT_RC_TIMEOUT	Position not reached
BAP_CHANGE_ALL_TO_DIST	No prism has been found during distance measurement with ATR, command changed from "All" to "Dist"
TMC_ACCURACY_GUARANTEE	Info, accuracy cannot be guaranteed
TMC_ANGLE_ACCURACY_ GUARANTEE	Info, only angle measurement valid, accuracy cannot be guaranteed
TMC_ANGLE_ERROR	Error, no valid angle measurement
TMC_ANGLE_NO_FULL_ CORRECTION	Warning, only angle measurement valid, accuracy cannot be guaranteed
TMC_ANGLE_OK	Warning, only angle measurement valid
TMC_BUSY	Error, TMC submodule already in use by another subsystem, command not processed
TMC_DIST_ERROR	An error occurred during distance measurement.
TMC_DIST_PPM	Error, wrong setting of PPM or MM on EDM
TMC_NO_FULL_CORRECTION	Warning, measurement without full correction

TMC_SIGNAL_ERROR	Error, no signal on EDM (only in signal mode)
RC_ABORT	Error, measurement aborted
RC_COM_TIMEDOUT	Error, communication timeout. (possibly increase COM timeout, see COM_SetTimeout)
RC_IVPARAM	Error, invalid DistMode
RC_SHUT_DOWN	Error, system stopped

See Also

-

Example

```
void MyMeasurement(BAP_MEASURE_PRG DistMode)
  {
  RC_TYPE          Result;
  BAP_MEASURE_PRG  DistMode;
  double           dHz, dV, dDist;

  DistMode = BAP_DEF_DIST
  Result = BAP_MeasDistanceAngle(DistMode,
                                 dHz, dV, dDist);
  if (rc != RC_OK)
    { // error-handling
    switch (rc)
      {
      case RC_IVPARAM:
      printf("Wrong value for DistMode!");
      break;

      case RC_ABORT:
      printf("Measurement aborted!");
      break;

      case RC_SHUT_DOWN:
      printf("System has been stopped!");
      break;

      case TMC_DIST_PPM:
      printf("PPM or MM should be switched off");
      printf(" when EDM is on -> no results!“);
      break;
```

```
      case TMC_DIST_ERROR:
      printf("Error occured during");
      printf(" distance measurement!");
      break;

      case TMC_ANGLE_ERROR:
      printf("Error occured while slope");
      printf(" was measured!");
      break;

      case TMC_BUSY:
      printf("TMC is busy!");
      break;

      case TMC_ANGLE_OK:
      printf("Angle without coordinates!")
      break;
      } // end of switch (rc)
    } // end of error handling
  else
    { // use results
    printf("horizontal angel [rad]: %d\n", dHz);
    printf("vertical angel [rad]  : %d\n", dV);
    printf("slopedistance [rad]   : %d\n", dDist);
    }
  } //end of MyMeasurement
```

8 BASIC MAN MACHINE INTERFACE - BMM

The subsystem BMM (Basic Man Machine Interface) implements the low-level functions for the MMI. These are also functions which are relevant for controlling the display, keyboard, character sets and the beeper (signalling device). Only beeper functions are implemented for now.

8.1 CONSTANTS AND TYPES

Constants for the signal-device

```
const short BMM_BEEP_STDINTENS = 100;
     // standard intensity of beep expressed as
     //a percentage
const short BMM_BEEP_STDFREQ   = 3900;
     // standard frequency of beep
```

8.2 FUNCTIONS

8.2.1 BMM_BeepAlarm - Output of an alarm-signal

C-Declaration

```
BMM_BeepAlarm(void)
```

VB-Declaration

```
VB_BMM_BeepAlarm()
```

ASCII-Request

```
%R1Q,11004:
```

ASCII-Response

```
%R1P,0,0:RC
```

Remarks

This function produces a triple beep with the configured intensity and frequency, which cannot be changed. If there is a continuous signal active, it will be stopped before.

Parameters

-

Return Codes

RC_OK	always

See Also

BMM_BeepNormal, BMM_BeepOn, BMM_BeepOff

8.2.2 BMM_BeepNormal - A single beep-signal

C-Declaration

```
BMM_BeepNormal(void)
```

VB-Declaration

```
VB_BMM_BeepNormal()
```

ASCII-Request

```
%R1Q,11003:
```

ASCII-Response

```
%R1P,0,0:RC
```

Remarks

This function produces a single beep with the configured intensity and frequency, which cannot be changed. If a continuous signal is active, it will be stopped first.

Parameters

-

Return Codes

RC_OK	always

See Also

BMM_BeepAlarm, BMM_BeepOn, BMM_BeepOff

8.2.3 BMM_BeepOn - Start a beep-signal

C-Declaration

```
BMM_BeepOn(short nIntens = BMM_BEEP_STDINTENS,
           short nFreq = BMM_BEEP_STDFREQ)
```

VB-Declaration

```
VB_BMM_BeepOn(ByVal nIntens As Integer,
              ByVal nFreq As Integer)
```

ASCII-Request

`%R1Q,11001:`*Volume[short], Frequency[short]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function switches on the beep-signal with the intensity `nIntens` and the frequency `nFreq`. If a continuous signal is active, it will be stopped first. Turn off the beeping device with `BMM_BeepOff`.

Parameters

`nIntens`	`in`	Intensity of the beep-signal (volume) expressed as a percentage. Default value is 100 %
`nFreq`	`in`	Frequency of the beep-signal. Default value is 3900 Hz. Range: 500 Hz ... 5000 Hz

Return Codes

`RC_OK`	always

See Also

`BMM_BeepOn, BMM_BeepAlarm, BMM_BeepNormal`

Example

```
BMM_BeepOn(BMM_BEEP_STDINTENS,
           BMM_BEEP_STDFREQ);

// wait for a second

BMM_BeepOff();
```

8.2.4 BMM_BeepOff - Stop active beep-signal

C-Declaration

```
BMM_BeepOff(void)
```

VB-Declaration

```
VB_BMM_BeepOff()
```

ASCII-Request

```
%R1Q,11002:
```

ASCII-Response

```
%R1P,0,0:RC
```

Remarks

-

Parameters

-

Return Codes

RC_OK	always

See Also

BMM_BeepOn, BMM_BeepAlarm, BMM_BeepNormal

Example

see BeepOn

9 COMMUNICATIONS - COM

This subsystem contains those functions, which are subsystem COM related, but will be executed as RPC's on the TPS1000 instrument. It provides a function to check communication between the computer and the TPS1000 and also some functions to get and set communication relevant parameters on the server side. Furthermore, it implements functions to switch on or off (sleep mode, shut down) the TPS1000 instrument.

9.1 CONSTANTS AND TYPES

Stop Mode

```
enum COM_TPS_STOP_MODE
{
  COM_TPS_SHUT_DOWN = 0,    // power down instrument
  COM_TPS_SLEEP     = 1     // puts instrument into sleep state
};
```

Start Mode

```
enum COM_TPS_STARTUP_MODE
{
  COM_TPS_REMOTE  = 1       // RPC's enabled, online mode
};
```

9.2 FUNCTIONS

9.2.1 COM_GetSWVersion - Retrieve Server Release Information

C-Declaration

```
COM_GetSWVersion(    short &nRel,
                     short &nVer,
                     short &nSubVer )
```

VB-Declaration

```
VB_COM_GetSWVersion( nRel     As Integer,
                     nVer     As Integer,
                     nSubVer  As Integer)
```

ASCII-Request

```
%R1Q,110:
```

ASCII-Response

`%R1P,0,0:`*RC, nDigits[short]*

Remarks

This function retrieves the current GeoCOM release (release, version and subversion) of the server.

Parameters

`nRel`	`out`	Software release.
`nVer`	`out`	Software version.
`nSubVer`	`out`	Software subversion (reserved).

Return Codes

`RC_OK`	On successful termination.

See Also

`CSV_GetSWVersion`
`COM_GetWinSWVersion`

Example

```
RC_TYPE   rc;
short     nRel, nSubVer, nVer;

COM_GetSWVersion(nRel, nVer, nSubVer);

printf(„TPS1000 GeoCOM Release:\n“);
printf(„Release      %02d\n“, nRel);
printf(„Version      %02d\n“, nVer);
printf(„Subversion   %02d\n“, nSubVer);
```

9.2.2 COM_SetSendDelay - Set Reply Delay

C-Declaration

`COM_SetSendDelay(short nSendDelay)`

VB-Declaration

`VB_COM_SetSendDelay(ByVal nSendDelay As Integer)`

ASCII-Request

`%R1Q,109:`*nSendDelay[short]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

The GeoCOM implementation of the server has been optimised for speed. If the server reacts to fast, then it may happen, that the client is not able to receive the reply (complete and) correctly. This RPC inserts a delay before the server responds to a request. This might be of interest especially for radio data links. Reset to no delay can be done with `nSendDelay = 0`.

Parameters

`nSendDelay`	`In`	Time of transmission delay in milliseconds.

Return Codes

`RC_OK`	On successful termination.

See Also

-

Example

```
RC_TYPE   rc;

rc = COM_SetSendDelay(5);
// do communication, long term RPC-calls
```

9.2.3 COM_Local - Switch TPS1000 into Local Mode

C-Declaration

`COM_Local(void)`

VB-Declaration

`VB_COM_Local()`

ASCII-Request

`%R1Q,1:`

ASCII-Response

`%R1P,0,0:`*`RC`*

Remarks

Leaves on-line mode and switches TPS1000 into local mode. If in local mode, no communication will take place. Any attempt of sending data will be ignored. Changing local into online mode can be done manually only.

Parameters

-

Return Codes

RC_OK	On successful termination.

See Also

COM_SwitchOffTPS

COM_SwitchOnTPS

Example

-

9.2.4 COM_SwitchOnTPS - Switch on TPS instrument

C-Declaration

```
COM_SwitchOnTPS(COM_TPS_STARTUP_MODE eOnMode)
```

VB-Declaration

```
VB_COM_SwitchOnTPS(ByVal eOnMode As Integer)
```

ASCII-Request

%R1Q,111:*eOnMode[short]*

ASCII-Response

```
%N1,0,255,,0%T0,0,0,:%R1P,0,0:0
```

Remarks

This function switches on the TPS1000 instrument and put it into remote mode. It can also be used to switch from sleep into remote mode.

Parameters

eOnMode	In	Run mode – use COM_TPS_REMOTE only! COM_TPS_LOCAL will yield to erroneous behaviour.

Return Codes

RC_OK	On successful termination.

See Also

COM_SwitchOffTPS

Example

-

9.2.5 COM_SwitchOffTPS - Switch off TPS1000 or Set Sleep Mode

C-Declaration

```
COM_SwitchOffTPS(COM_TPS_STOP_MODE eOffMode)
```

VB-Declaration

```
VB_COM_SwitchOffTPS(ByVal eOffMode As Integer)
```

ASCII-Request

`%R1Q,112:`*eOffMode[short]*

ASCII-Response

if RPC was successful and sign-off is enabled then

either `%N1,0,255,,0%T0,0,0,:%R1P,1,0:0,1` (sleep)

or `%N1,0,255,,0%T0,0,0,:%R1P,1,0:0,0` (shut down)

optionally followed by

`%R1P,0,0:`*RC*

else

`%R1P,0,0:`*RC*

Remarks

This function switches off the TPS1000 instrument or put it into sleep mode.

Parameters

`eOffMode`	`In`	Stop mode - see Constants and Types.

Return Codes

`RC_OK`	On successful termination.
`RC_COM_SRVR_IS_SLEEPING`	TPS has gone to sleep. Wait and try again. Only if called repetitive with `eOffMode == COM_TPS_SLEEP` and sign-off is enabled.

See Also

`COM_SwitchOnTPS`

Example

-

9.2.6 COM_NullProc – Check Communication

C-Declaration

```
COM_NullProc(void)
```

VB-Declaration

```
VB_COM_NullProc()
```

ASCII-Request

```
%R1Q,0:
```

ASCII-Response

```
%R1P,0,0:RC
```

Remarks

This function does not provide any functionality except of checking if the communication is up and running.

Parameters

-

Return Codes

RC_OK	On successful termination.

See Also

-

Example

-

9.2.7 COM_EnableSignOff – Enable Remote Mode Logging

C-Declaration

```
COM_EnableSignOff(BOOLE bEnable)
```

VB-Declaration

```
VB_COM_EnableSignOff(ByVal bEnable As Integer)
```

ASCII-Request

```
%R1Q,115:bEnable[Boolean]
```

ASCII-Response

```
%R1P,0,0:RC
```

Remarks

This function enables logging if the Remote mode changes. See also section **3.5 TPS1000 Instrument Modes of Operation** for further explanations.

Note: The flag will be reset each time the instrument turns on.

Parameters

bEnable	In	TRUE: enable mode logging

Return Codes

RC_OK	On successful termination.

See Also

COM_GetTPSState

Example

-

9.2.8 COM_GetBinaryAvailable – Get Binary Attribute of Server

C-Declaration

```
COM_GetBinaryAvailable(BOOLE &bAvailable)
```

VB-Declaration

```
VB_COM_GetBinaryAvailable(bAvailable As Integer)
```

ASCII-Request

```
%R1Q,113:
```

ASCII-Response

```
%R1P,0,0:RC, bAvailable[Boolean]
```

Remarks

This function gets the ability information about the server to handle binary communication. Since TPS1000 Release 2.20 the client may make requests in binary format which speeds up the communication by about 40-50%.

Parameters

bAvailable	Out	TRUE: binary operation enabled. FALSE: ASCII operation enabled.

Return Codes

RC_OK	On successful termination.

See Also

COM_SetBinaryAvailable
COM_SetFormat
COM_GetFormat

9.2.9 COM_SetBinaryAvailable – Set Binary Attribute of Server

C-Declaration

```
COM_SetBinaryAvailable(BOOLE bAvailable)
```

VB-Declaration

```
VB_COM_SetBinaryAvailable(ByVal bAvailable As Integer)
```

ASCII-Request

%R1Q,114:*bAvailable[Boolean]*

ASCII-Response

%R1P,0,0:*RC*

Remarks

This function sets the ability of the server to handle binary communication. With this function, one can force to communicate in ASCII only. During initialisation, the client checks if binary communication is enabled / possible or not which depends on this flag.

Parameters

bAvailable	In	TRUE: enable binary operation. FALSE: enable ASCII operation only.

Return Codes

RC_OK	On successful termination.

See Also

COM_GetBinaryAvailable
COM_SetFormat

Example

-

10 CENTRAL SERVICES - CSV

The subsystem Central Services implements some centralised functions to maintain global data of the TPS system software. Examples are date and time or the instrument's name.

10.1 USAGE

These functions do not depend on other subsystems. Since this part is responsible for global data, any function can be called at any time. Some values do not have pre-set values (e.g. the user name), hence calling the get function without having set the value previously will yield in an error return code.

10.2 CONSTANTS AND TYPES

TPS Device Configuration Type

```
struct  TPS_DEVICE
{
  TPS_DEVICE_CLASS Class; // device precision class
  TPS_DEVICE_TYPE  Type;  // device configuration type
};
```

TPS Device Precision Class

```
enum TPS_DEVICE_CLASS
{
  TPS_CLASS_1100 = 0,     // TPS1000 family member,
                          // 1 mgon, 3"
  TPS_CLASS_1700 = 1,     // TPS1000 family member,
                          // 0.5 mgon, 1.5"
  TPS_CLASS_1800 = 2,     // TPS1000 family member,
                          // 0.3 mgon, 1"
  TPS_CLASS_5000 = 3,     // TPS2000 family member
  TPS_CLASS_6000 = 4,     // TPS2000 family member
  TPS_CLASS_1500 = 5      // TPS1000 family member
};
```

TPS Device Configuration Type

```
enum TPS_DEVICE_TYPE
{
  TPS_DEVICE_T   = 0x00000,  // theodolite without
                             // built-in EDM
  TPS_DEVICE_TC1 = 0x00001,  // tachymeter
                             // (older EDM built-in)
  TPS_DEVICE_TC2 = 0x00002,  // tachymeter
                             // (newer EDM built-in)
  TPS_DEVICE_MOT = 0x00004,  // motorized device
  TPS_DEVICE_ATR = 0x00008,  // automatic target
                             // recognition
  TPS_DEVICE_EGL = 0x00010,  // electronic guide light
  TPS_DEVICE_DB  = 0x00020,  // reserved
  TPS_DEVICE_DL  = 0x00040,  // diode laser
  TPS_DEVICE_LP  = 0x00080,  // laser plummet
  TPS_DEVICE_SIM = 0x04000   // runs on simulation,
                             // not on hardware
};
```

General Date and Time

```
struct  DATIME {
  DATE_TYPE   Date;
  TIME_TYPE   Time;
};
```

General Date

```
struct DATE_TYPE {
  short   Year;           // year
  BYTE    Month;          // month in year 1..12
  BYTE    Day;            // day in month 1..31
};
```

General Time

```
struct TIME_TYPE {
  BYTE    Hour;           // 24 hour per day  0..23
  BYTE    Minute;         // minute 0..59
  BYTE    Second;         // seconds 0..59
};
```

10.3 FUNCTIONS

10.3.1 CSV_GetInstrumentNo - Get factory defined instrument number

C-Declaration

```
CSV_GetInstrumentNo(long &SerialNo)
```

VB-Declaration

```
VB_CSV_GetInstrumentNo(SerialNo As Long)
```

ASCII-Request

```
%R1Q,5003:
```

ASCII-Response

`%R1P,0,0:`*RC, SerialNo[long]*

Remarks

-

Parameters

`SerialNo`	out	The serial number.

Return-Codes

`RC_OK`	Execution successful.
`RC_UNDEFINED`	Instrument number not yet set.

Example

```
RC_TYPE   rc;
long      SerialNo;

rc = CSV_GetInstrumentNo(SerialNo);
if (rc == RC_OK)
{
     // use SerialNo
}
else
{
     // instrument number not yet set
}
```

10.3.2 CSV_GetInstrumentName - Get Leica specific instrument name

C-Declaration

```
CSV_GetInstrumentName(char *Name)
```

VB-Declaration

```
VB_CSV_GetInstrumentName(Name As String)
```

ASCII-Request

```
%R1Q,5004:
```

ASCII-Response

```
%R1P,0,0:RC,Name[string]
```

Remarks

-

Parameters

Name	out	The instrument name

Return-Codes

RC_OK	Execution successful.
RC_UNDEFINED	Instrument name not set yet.

Example

```
RC_TYPE   rc;

rc = CSV_GetInstrumentName(szName);
if (rc == RC_OK)
{
  // use instrument name
}
else
{
  // instrument name not set yet
  // (incomplete calibration data)
}
```

10.3.3 CSV_GetUserInstrumentName - Get user defined instrument name

C-Declaration

```
CSV_GetUserInstrumentName(char *Name)
```

VB-Declaration

```
VB_CSV_GetUserInstrumentName(Name As String)
```

ASCII-Request

```
%R1Q,5006:
```

ASCII-Response

```
%R1P,0,0:RC,Name[string]
```

Remarks

This name can be set by the user (see `CSV_SetUserInstrumentName`) If no user instrument name is set the return code is RC_UNDEFINED and the Leica specific instrument name is returned.

Parameters

`Name`	out	The user defined instrument name.

Return-Codes

`RC_OK`	Execution successful.
`RC_UNDEFINED`	User instrument name not set yet. The Leica specific instrument name is returned instead of the user instrument name.

See Also

`CSV_SetUserInstrumentName`

Example

```
RC_TYPE    rc;
char *     name = "                    ";

rc = CSV_GetUserInstrumentName(Name);
if (rc == RC_OK)
{
     // the user instrument is already set and
     // returned
}
else
{
     // the user instrument name is not already
     // set; the Leica specific instrument name is
     // returned instead of.
}
```

10.3.4 CSV_SetUserInstrumentName - Set user defined instrument name

C-Declaration

```
CSV_SetUserInstrumentName(char *Name)
```

VB-Declaration

```
VB_CSV_SetUserInstrumentName(ByVal Name As String)
```

ASCII-Request

`%R1Q,5005:`*Name[string]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

-

Parameters

`Name`	in	The user defined instrument name.

Return-Codes

`RC_OK`	Execution always successful.

See Also

`CSV_GetUserInstrumentName`

Example

```
RC_TYPE   rc;
char *    Name = "New name";
rc = CSV_SetUserInstrumentName(Name);
```

10.3.5 CSV_GetDeviceConfig - Get instrument configuration

C-Declaration

```
CSV_GetDeviceConfig(TPS_DEVICE &Device);
```

VB-Declaration

```
VB_CSV_GetDeviceConfig(Device As TPS_DEVICE)
```

ASCII-Request

`%R1Q,5035:`

ASCII-Response

`%R1P,0,0:`*RC, DevicePrecisionClass[long], DeviceConfigurationType[long]*

Remarks

This function returns information about the class and the configuration type of the instrument.

Parameters

`Device`	out	System information (see data type description for further information).

Return-Codes

`RC_OK`	Well configured.
`RC_UNDEFINED`	Instrument precision class undefined.

Example

```
RC_TYPE      rc;
TPS_DEVICE   Device;

rc = CSV_GetDeviceConfig(Device);
if (rc == RC_OK)
{
    // Use system information
}
else
{
    // Intrument precision class undefined
    // (incomplete calibration data)
}
```

10.3.6 CSV_GetDateTime- Get date and time.

C-Declaration

```
CSV_GetDateTime(DATIME &DateAndTime)
```

VB-Declaration

```
VB_CSV_GetDateTime (DateAndTime As DATIME)
```

ASCII-Request

```
%R1Q,5008:
```

ASCII-Response

`%R1P,0,0:`*RC,Year[short],Month,Day,Hour,Minute,Second[all byte]*

Remarks

The ASCII response is formatted corresponding to the data type `DATIME`. A possible response can look like this: `%R1P,0,0:0,1996,'07','19','10','13','2f'` (see chapter ASCII data type declaration for further information)

Parameters

`DateAndTime`	out	Encoded date and time.

Return-Codes

`RC_OK`	Execution successful.
`RC_UNDEFINED`	Time and/or date are not set (yet).

See Also

`CSV_SetDateTime`

Example

```
RC_TYPE   rc;
DATIME    DateAndTime;

rc = CSV_GetDateTime(DateAndTime);
if (rc == RC_OK)
{
    // use Date and time
}
else
{
    // time and/or date is not set (yet)
    // use CSV_SetDateTime to set date and time
    // (March 25 1997, 10:20)
    DateAndTime.Date.Year   = 1997;
    DateAndTime.Date.Month  = 3;
    DateAndTime.Date.Day    = 25;
    DateAndTime.Time.Hour   = 10;
    DateAndTime.Time.Minute = 20;
    DateAndTime.Time.Second = 0;
    rc = CSV_SetDateTime(DateAndTime);
}
```

10.3.7 CSV_SetDateTime - Set date and time

C-Declaration

```
CSV_SetDateTime(DATIME DateAndTime)
```

VB-Declaration

```
VB_CSV_SetDateTime(ByVal DateAndTime As DATIME)
```

ASCII-Request

`%R1Q,5007:`*Year[short],Month,Day,Hour,Minute,Second[all byte]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

It is not possible to set invalid date or time. See data type description of `DATIME` for valid date and time.

Parameters

`DateAndTime`	in	Encoded date and time.

Return-Codes

`RC_OK`	Execution always successful.

See Also

`CSV_GetDateTime`

Example

See `CSV_GetDateTime`.

10.3.8 CSV_GetSWVersion - Get Software Version

C-Declaration

```
CSV_GetSWVersion(short &nRelease, short &nVersion,
                 short &nSubVersion)
```

VB-Declaration

```
VB_COM_GetSWVersion(nRelease As Integer,
                    nVersion As Integer,
                    nSubVersion As Integer)
```

ASCII-Request

```
%R1Q,5034:
```

ASCII-Response

`%R1P,0,0:`*RC,nRelease,nVersion,nSubVersion[all short]*

Remarks

Returns the system software version.

Parameters

`nRelease`	out	Release
`nVersion`	out	Version
`nSubVersion`	out	Sub Version

Return-Codes

`RC_OK`	Execution always successful.

Example

```
RC_TYPE  rc;
short    nRel, nVers, nSubVers;
char     szBuffer[17]

rc = CSV_GetSWVersion(nRel, nVers, nSubVers)
sprintf(szBuffer, "Version %02d.%02d.%02d",
        nRel, nVers, nSubVers);
Returns: nRel = 2, nVers = 20, nSubVers = 0
        szBuffer = “Version 02.20.00”
```

10.3.9 CSV_GetVBat - Get the value of the voltage supply

C-Declaration

```
CSV_GetVBat(double &VBat)
```

VB-Declaration

```
VB_CSV_GetVBat(VBat As double)
```

ASCII-Request

`%R1Q,5009:`

ASCII-Response

`%R1P,0,0:`*RC,Vbat[double]*

Remarks

The value of `Vbat` gives information about the state of charge of the battery.

value of Vbat	Charge State
12,7 V < Vbat	full
12,4 V < Vbat < 12,7 V	near full
11,1 V < Vbat < 12,4 V	good
10,5 V < Vbat < 11,1 V	empty
Vbat V < 10,5	powered off

Parameters

Vbat	out	Voltage of the battery [V]. The voltage has an exactness of ±0.2V.

Return-Codes

RC_OK	Execution always successful.

Example

```
RC_TYPE   rc;
double    VBat;

rc = CSV_GetVBat(VBat);
if (Vbat < 11.1)
{
     // Warning: battery empty
}
```

10.3.10 CSV_GetVMem - Get value of the memory backup voltage supply

C-Declaration

```
CSV_GetVMem(double &VMem)
```

VB-Declaration

```
VB_CSV_GetVMem(VMem As double)
```

ASCII-Request

```
%R1Q,5010:
```

ASCII-Response

```
%R1P,0,0:RC,VMem[double]
```

Remarks

A value of `Vmem` > 3.1 V means OK.

Parameters

Vmem	out	Voltage of the memory backup system [V]. The voltage has an exactness of ±0.1V.

Return-Codes

RC_OK	Execution always successful.

Example

```
RC_TYPE   rc;
double    VMem;

rc = CSV_GetVMem(VMem);
if (Vbat > 3.1)
{
    // Message: memory backup voltage OK
}
else
{
    // Warning: memory backup voltage not OK
    // exchange battery it in Leica service
}
```

10.3.11 CSV_GetIntTemp - Get the temperature

C-Declaration

```
CSV_GetIntTemp(double &Temp)
```

VB-Declaration

```
VB_CSV_GetIntTemp(Temp As double)
```

ASCII-Request

```
%R1Q,5011:
```

ASCII-Response

```
%R1P,0,0:RC,Temp[long]
```

Remarks

Get the internal temperature of the instrument, measured on the Mainboard side. Values are reported in degrees Celsius.

Parameters

Temp	out	Instrument temperature.

Return-Codes

RC_OK	Execution always successful.

Example

```
RC_TYPE   rc;
double    Temp;

rc = CSV_GetIntTemp(Temp);
// use temperature information
```

11 CONTROLLER TASK – CTL

This chapter describes one RPC only, which shows how often the TPS1000 instrument has been switched on and how often the instrument fell asleep.

11.1 FUNCTIONS

11.1.1 CTL_GetUpCounter – Get Up Counter

C-Declaration

```
CTL_GetUpCounter( short &nPowerOn,
                  short &nWakeUp )
```

VB-Declaration

```
VB_CTL_GetUpCounter( nPowerOn As Integer,
                     nWakeUp  As Integer )
```

ASCII-Request

```
%R1Q,12003:
```

ASCII-Response

`%R1P,0,0:`*RC, nPowerOn[short] , nWakeUp[short]*

Remarks

This function retrieves how often, since the last call of this function, the TPS1000 instrument has been switched on and how often it has been awakened from sleep mode. Both counters are unique and will be reset to Zero once the function has been called.

Parameters

`nPowerOn`	`out`	Switch on counter.
`nWakeUp`	`out`	Waken up counter.

Return Codes

`RC_OK`	On successful termination.

See Also

`COM_GetTPSState`
`COM_SwitchOffTPS`

Example

```
RC_TYPE    RetCode;
short      nPowerOn, nWakeUp;

// do some stuff

RetCode = CTL_GetUpCounter(nPowerOn, nWakeUp);
if (RetCode != RC_OK)
{
  // handle error
}

if (nPowerOn > 0)
{
  // instrument has been switched off in between
}

if (nWakeUp > 0)
{
  // instrument has been fallen asleep in between
}
...
```

12 ELECTRONIC DISTANCE MEASUREMENT - EDM

The subsystem electronic distance measurement (EDM) is the adaptation of the distance measurement devices on the Theodolite. These devices can be integrated in the whole system or added to the Theodolite.

With the functionality of EDM, one can switch on or off the EDM, the boomerang filter, the Laserpointer and the Tracklight respectively. Additionally it is possible to tune the brightness of the Tracklight.

12.1 USAGE

It is important to initialise EDM before start working.

In order to use the functions concerning the Laserpointer, boomerang filter and Tracklight, make sure these devices are available. If not, these functions return error messages.

The Laserpointer is available for EDM of type DIOR3002S or DISTO.

The boomerang filter is available for Theodolite of type TC1600, TC2002 and also of type TCW II.

12.2 CONSTANTS AND TYPES

On/off switch

```
enum ON_OFF_TYPE // on/off switch type
    {
    OFF = 0,
    ON  = 1
    };
```

Intensity of Tracklight

```
typedef enum EDM_TRKLIGHT_BRIGHTNESS
    {
    EDM_LOW_BRIGHTNESS    = 0,
    EDM_MEDIUM_BRIGHTNESS = 1,
    EDM_HIGH_BRIGHTNESS   = 2
    };
```

12.3 FUNCTIONS

12.3.1 EDM_Laserpointer - Switch on/off laserpointer

C-Declaration

`EDM_Laserpointer(ON_OFF_TYPE eLaser)`

VB-Declaration

`VB_EDM_Laserpointer(ByVal eLaser As Integer)`

ASCII-Request

`%R1Q,1004:`*eLaser[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

Subsystem EDM has to be of type DIOR3002S or DISTO.

If EDM is switched off by the Auto-Power-Off-Function, the Laserpointer possibly cannot be activated again. In this case deactivate the Laserpointer with `EDM_Laserpointer(OFF)` and activate it again.

Parameters

`eOn`	`in`	`ON` - switch Laserpointer on `OFF` - switch Laserpointer off

Return Codes

`RC_OK`	Laserpointer switched on/off
`RC_NOT_IMPL`	Laserpointer not available or implemented
`EDM_RC_HWFAILURE`	hardware error in EDM occurred
`EDM_COMERR`	error on communication with EDM
`RC_TIME_OUT`	process time out
`RC_ABORT`	function interrupted
`RC_SYSBUSY`	EDM already busy
`RC_IVPARAM`	parameter value must be `ON` or `OFF`
`RC_UNDEFINED`	instrument name could not be read

See Also

-

Example

```
RC_TYPE    rc;

// switch on laserpointer
rc = EDM_Laserpointer(ON);

if (rc != RC_OK)
  { // error-handling
  switch (rc)
    {
    case RC_NOT_IMPL:
      printf("This function is not implemented
              or available. Laserpointer is
              available for EDM of type
              DIOR3002S or DISTO.");
      break;

    case EDM_RC_HWFAILURE:
      printf("Hardware error occured in EDM!");
      break;

    case EDM_COMERR:
      printf("Error on communication with EDM!");
      break;

    case RC_TIME_OUT:
      printf("Process time out");
      break;

    case RC_ABORT:
      printf("Function aborted!");
      break;

    case RC_UNDEFINED:
      printf("Instrument name is not set!");
      printf("Fatal error: call service!");
      break;

    case RC_SYSBUSY:
      printf("EDM is already busy!");
      break;
```

```
      case RC_IVPARAM:
        printf("Parameter of EDM_Laserpointer
                must be of ON_OFF_TYPE!");
        break;
      } // end of switch (rc)
    }  // end of error handling
  else if (rc == RC_OK)
    {
    // use laserpointer
    }
```

12.3.2 EDM_On - Switch on/off EDM

C-Declaration

`EDM_On(ON_OFF_TYPE eOn)`

VB-Declaration

`VB_EDM_On(ByVal eOn As Integer)`

ASCII-Request

`%R1Q,1010:`*eOn[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

Normal distance measurement switches on or off the EDM automatically. If there is no supply voltage for the EDM, `EDM_On` also connects it. Normally the supply voltage is switched off together with the EDM, unless the Tracklight or the diode laser is switched on too, or the permanent power-on-mode is set. These functions overlie the functionality of `EDM_On` and EDM cannot be switched off before the Tracklight or the diode laser stops working. However, after 10 min the EDM will be switched off.

Parameters

`eOn`	`in`	`ON` - switch EDM on `OFF` - switch EDM off

Return Codes

`RC_OK`	EDM switched on/off
`RC_SYSBUSY`	EDM already busy

EDM_COMERR	communication with EDM does not work correctly
EDM_ERR12	supply voltage to low
EDM_RC_HWFAILURE	hardware error occurred
RC_TIME_OUT	process time out
RC_ABORT	function aborted
RC_UNDEFINED	instrument name is undefined

See Also

-

Example

For more return codes see also example at chapter 12.3.1 "**EDM_Laserpointer**".

```
RC_TYPE    rc;

// switch on EDM
rc = EDM_On(ON);

if (rc != RC_OK)
  { // error-handling
  switch (rc)
    {
    case EDM_ERR12:
      printf("Supply voltage too low!");
      break;

    case ....
      ...
    } // end of switch (rc)
  }  // end of error handling
else if (rc == RC_OK)
  {
  //use EDM
  }
```

12.3.3 EDM_GetBumerang - Get status of boomerang filter

C-Declaration

```
EDM_GetBumerang(ON_OFF_TYPE &eBoomerangFilter)
```

VB-Declaration

```
VB_EDM_GetBumerang(eBoomerangFilter As Integer)
```

ASCII-Request

```
%R1Q,1044:
```

ASCII-Response

```
%R1P,0,0:RC,eBumerangFilter[long]
```

Remarks

Call this function to retrieve the status of the "boomerang filter" (i.e. `ON, OFF`). If the distance is within 60-100 meters and the boomerang filter is turned off, wrong measurement results are possible. With the boomerang filter turned on, the measurement accuracy can be increased until up to max. 2.5 cm.
The boomerang filter is not available for some add-on EDM's.

Parameters

`eOn`	`out`	`ON` - boomerang filter on `OFF` -boomerang filter off

Return Codes

`RC_OK`	boomerang filter switched on/off
`RC_IVRESULT`	error occurred, wrong result
`RC_SYSBUSY`	EDM already busy
`RC_NOT_IMPL`	boomerang filter not available, see remarks
`EDM_COMERR`	communication with EDM does not work correctly
`EDM_RC_HW_FAILURE`	hardware error occurred
`RC_TIME_OUT`	process time out
`RC_ABORT`	function aborted
`RC_UNDEFINED`	instrument name is undefined
`EDM_ERR12`	supply voltage to low

See Also

`EDM_SetBumerang()`

Example

For more return codes see also example at chapter 12.3.1 "**EDM_Laserpointer**".

```
RC_TYPE      rc;
ON_OFF_TYPE  eBumerang;

// is bumerang filter on?
rc = EDM_GetBumerang(eBumerang);

if (rc != RC_OK)
  { // error-handling
  switch (rc)
    {
    case EDM_ERR12:
      printf("Supply voltage too low!");
      break;

    case RC_NOT_IMPL:
      printf("Bumerangfilter not availabe or
              function not implemented!
              Bumerangfilter is available for
              TC1600, TC2002 and TCW II.");
      break;

    case RC_IVRESULT:
      printf("Error occured, wrong result
              returned!");
      break;

    case ....
      ...
    } // end of switch (rc)
  }  // end of error handling
```

12.3.4 EDM_SetBumerang - Switch boomerang filter on/off

C-Declaration

```
EDM_SetBumerang(ON_OFF_TYPE eOn)
```

VB-Declaration

```
VB_EDM_SetBumerang(ByVal eOn As Integer)
```

ASCII-Request

`%R1Q,1007:`*eOn[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

The boomerang filter is not available for some add-on EDM's.

Parameters

`eOn`	`in`	`ON` - switch boomerang filter on `OFF` - switch boomerang filter off

Return Codes

`RC_OK`	boomerang filter switched on/off
`RC_IVRESULT`	error occurred, wrong result
`RC_SYSBUSY`	EDM already busy
`RC_NOT_IMPL`	boomerang filter not available, see remarks
`RC_IVPARAM`	parameter value must be `ON` or `OFF`
`EDM_COMERR`	communication with EDM does not work correctly
`EDM_ERR12`	supply voltage to low
`EDM_RC_HW_FAILURE`	hardware error occurred in EDM
`RC_TIME_OUT`	process time out
`RC_ABORT`	function aborted
`RC_UNDEFINED`	instrument name is undefined

See Also

`EDM_GetBumerang()`

Example

See preceding examples.

12.3.5 EDM_GetTrkLightBrightness - Get value of intensity of tracklight

C-Declaration

```
EDM_GetTrkLightBightness(EDM_TRKLIGHT_BRIGHTNESS
                         &eBrightness)
```

VB-Declaration

```
VB_EDM_GetTrkLightBightness(eBrightness As Integer)
```

ASCII-Request

```
%R1Q,1041:
```

ASCII-Response

```
%R1Q,0,0:RC,eBrightness[long]
```

Remarks

The Tracklight must be available for EDM.

Parameters

eBrightness	out	EDM_LOW_BRIGHTNESS
		EDM_MEDIUM_BRIGHTNESS
		EDM_HIGH_BRIGHTNESS

Return Codes

RC_OK	value of Tracklight brightness returned
RC_NOT_IMPL	Tracklight not available, see remarks

See Also

EDM_SetTrkLightBrightness()

Example

See preceding examples.

12.3.6 EDM_SetTrkLightBrightness - Change intensity of tracklight

C-Declaration

```
EDM_SetTrkLightBightness(EDM_TRKLIGHT_BRIGHTNESS
                         eBrightness)
```

VB-Declaration

```
VB_EDM_SetTrkLightBightness(ByVal eBrightness As
                            Integer)
```

ASCII-Request

`%R1Q,1032:`*eBrightness[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

The Tracklight must be available for EDM.

Parameters

`eBrightness`	`in`	`EDM_LOW_BRIGHTNESS`
		`EDM_MEDIUM_BRIGHTNESS`
		`EDM_HIGH_BRIGHTNESS`

Return Codes

`RC_OK`	Tracklight brightness is set
`RC_SYSBUSY`	EDM already busy
`RC_NOT_IMPL`	Tracklight not available, see remarks

See Also

`EDM_GetTrkLightBrightness()`

Example

See preceding examples.

12.3.7 EDM_GetTrkLightSwitch - Get status of tracklight switch

C-Declaration

```
EDM_GetTrkLightSwitch(ON_OFF_TYPE &eRunnig)
```

VB-Declaration

```
VB_EDM_GetTrkLightSwitch(eRunnig As Integer)
```

ASCII-Request

`%R1Q,1040:`

ASCII-Response

`%R1P,0,0:`*RC*`,`*eBumerang[long]*

Remarks

The Tracklight must be available for EDM.

Parameters

eRunning	out	ON - Tracklight is on OFF - Tracklight is off

Return Codes

RC_OK	Tracklight is on/off
RC_NOT_IMPL	Tracklight not available, see remarks

See Also

EDM_SetTrkLightSwitch()

Example

See preceding examples.

12.3.8 EDM_SetTrkLightSwitch - Switch on/off tracklight

C-Declaration

```
EDM_SetTrkLightSwitch(ON_OFF_TYPE eRunnig)
```

VB-Declaration

```
VB_EDM_SetTrkLightSwitch(ByVal eRunnig As Integer)
```

ASCII-Request

%R1Q,1031:*eRunning[long]*

ASCII-Response

%R1P,0,0:*RC*

Remarks

The Tracklight must be available for EDM.

Parameters

eRunning	in	ON - switch on Tracklight OFF - switch off Tracklight

Return Codes

RC_OK	Tracklight switched on/off
RC_IVRESULT	error occurred, wrong result
RC_SYSBUSY	EDM already busy
RC_NOT_IMPL	Tracklight not available, see remarks

See Also

`EDM_GetTrkLightSwitch()`

Example

See preceding examples.

13 MOTORISATION - MOT

The subsystem ‘Motorisation’ controls the motorised drive of the axis.

13.1 USAGE

Within the subsystem, there exist three different types of functions:

"Open-End" functions: These functions start a motorisation control task and continue execution until cancellation. Special control functions are used to cancel such functions. An example for this type of function is the speed control function `MOT_SetVelocity`.

"Terminating" functions: These functions start control tasks, which terminate automatically. Examples for this type are positioning functions for example `MOT_StartController` and `MOT_StopController`.

Functions for the parameter handling: These functions manage system parameters. Examples are control parameter, motion parameter, tolerance and system configuration parameters (Example: `MOT_ReadLockStatus`).

13.2 CONSTANTS AND TYPES

Lock Conditions

```
enum MOT_LOCK_STATUS
{
  MOT_LOCKED_OUT = 0,    // locked out
  MOT_LOCKED_IN  = 1,    // locked in
  MOT_PREDICTION = 2     // prediction mode
};
```

Controller Stop Mode

```
enum MOT_STOPMODE
{
  MOT_NORMAL   = 0,      // slow down with current acceleration
  MOT_SHUTDOWN = 1       // slow down by switch off power supply
};
```

Values for Horizontal (instrument) and Vertical (telescope) Speed

```
struct MOT_COM_PAIR
{
  double  adValue[MOT_AXES];
};
```

Controller Configuration

```
enum MOT_MODE
{
  MOT_POSIT     = 0,   // configured for relative postioning
  MOT_OCONST    = 1,   // configured for constant speed
  MOT_MANUPOS   = 2,   // configured for manual positioning
                       // default setting
  MOT_LOCK      = 3,   // configured as "Lock-In"-controller
  MOT_BREAK     = 4,   // configured as "Brake"-controller
                       // do not use 5 and 6
  MOT_TERM      = 7    // terminates the controller task
};
```

Number of axis

```
const short MOT_AXES = 2;
```

13.3 FUNCTIONS

13.3.1 MOT_ReadLockStatus - Return condition of LockIn control

C-Declaration

```
MOT_ReadLockStatus(MOT_LOCK_STATUS &Status)
```

VB-Declaration

```
VB_MOT_ReadLockStatus(Status As Integer)
```

ASCII-Request

```
%R1Q,6021:
```

ASCII-Response

```
%R1P,0,0:RC,Status[long]
```

Remarks

This function returns the current condition of the LockIn control (see subsystem AUT for further information). This command is valid for TCA instruments only.

Parameters

`Status`	`out`	lock information

Return-Codes

`RC_OK`	Execution successful.
`RC_NOT_IMPL`	No motorisation available (no TCA instrument).

Example

```
RC_TYPE          rc;
MOT_LOCK_STATUS  Status;

rc = MOT_ReadLockStatus(Status)
if (rc == RC_OK)
{
  // use lock status information
}
else
{
  // this is no TCA instrument
}
```

13.3.2 MOT_StartController - Start motor controller

C-Declaration

```
MOT_StartController(MOT_MODE ControlMode)
```

VB-Declaration

```
VB_MOT_StartController(ControlMode As Integer)
```

ASCII-Request

`%R1Q,6001:`*ControlMode[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This command is used to enable remote or user interaction to the motor controller.

Parameters

ControlMode	in	Controller mode. If used together with MOT_SetVelocity the control mode has to be.

Return-Codes

RC_OK	Execution successful.
RC_IVPARAM	The value of ControlMode is not MOT_OCONST.
RC_NOT_IMPL	No motorization available (no TCA instrument).
MOT_RC_BUSY	Subsystem is busy (e.g. controller already started).
MOT_RC_UNREADY	Subsystem is not initialised.

See Also

MOT_SetVelocity
MOT_StopController

Example

see MOT_SetVelocity

13.3.3 MOT_StopController - Stop motor controller

C-Declaration

```
MOT_StopController(MOT_STOPMODE Mode)
```

VB-Declaration

```
VB_MOT_StopController(Mode As Integer)
```

ASCII-Request

%R1Q,6002:*Mode[long]*

ASCII-Response

%R1P,0,0:*RC*

Remarks

This command is used to stop movement and to stop the motor controller operation.

Parameters

Mode	in	Stop mode

Return-Codes

RC_OK	Execution successful.

MOT_RC_NOT_BUSY	No movement in progress (e.g. stop without start).

See Also

MOT_SetVelocity
MOT_StartController
AUT_SetLockStatus

Example

see MOT_SetVelocity

13.3.4 MOT_SetVelocity - Drive Instrument with visual control

C-Declaration

```
MOT_SetVelocity(MOT_COM_PAIR RefOmega)
```

VB-Declaration

```
VB_MOT_SetVelocity Lib(RefOmega As MOT_COM_PAIR)
```

ASCII-Request

%R1Q,6004:*HZ-Speed[double],V-Speed[double]*

ASCII-Response

%R1P,0,0:*RC*

Remarks

This command is used to set up the velocity of motorization. This function is valid only if MOT_StartController(MOT_OCONST) has been called previously. RefOmega[0] denotes the horizontal and RefOmega[1] denotes the vertical velocity setting.

Parameters

RefOmega	in	The speed in horizontal and vertical direction in rad/s. The maximum speed is +/- 0.79 rad/s each.

Return-Codes

RC_OK	Execution successful
RC_IVPARAM	RefOmega.adValue[HZ] and/or RefOmega.adValue[V] values are not within the boundaries.
MOT_RC_NOT_CONFIG	System is not in state MOT_CONFIG or MOT_BUSY_OPEN_END (e.g. missing ‘start controller’).

MOT_RC_NOT_OCONST	Drive is not in mode MOT_OCONST (set by MOT_StartController).
RC_NOT_IMPL	No motorization available (no TCA instrument).

See Also

MOT_StartController
MOT_StopController
AUT_SetLockStatus

Example

```
RC_TYPE         rc;
MOT_COM_PAIR    RefOmega;

// set parameter
RefOmega.adValue[0] = 0.05;
RefOmega.adValue[1] = 0.05;

// stop controller and any possible movements
(void) MOT_StopController(MOT_NORMAL);
// wait at least 5 sec.
wait(5);

// start controller; the only valid mode
// for SetVelocity is MOD_OCONST
rc = MOT_StartController(MOT_OCONST);
if (rc == RC_OK)
{
  rc = MOT_SetVelocity(RefOmega);
  // insert here a time delay or a wait for user
  // action; the movement stops by calling
  // MOT_StopController
}
// stop controller and movements abruptly
rc = MOT_StopController(MOT_SHUTDOWN);

// restart controller with default setting
rc = MOT_StartController(MOT_MANUPOS);
if (rc != RC_OK)
{
  // handle error
}
```

14 SUPERVISOR - SUP

The subsystem ‘Supervisor’ performs the continuous control of the system (e.g. battery voltage, temperature) and allows to display automatically status information (e.g. system time, battery-, position-, Memory-Card-, and inclination measurement icons as well as local-remote display). It also controls the automatic shutdown mechanism.

14.1 USAGE

14.2 CONSTANTS AND TYPES

On/Off Switch

```
enum ON_OFF_TYPE
{
  OFF = 0,
  ON  = 1
};
```

Automatic Shutdown Mechanism for the System

```
enum SUP_AUTO_POWER
{
  AUTO_POWER_DISABLED = 0,  // deactivate the mechanism
  AUTO_POWER_SLEEP    = 1,  // activate sleep mechanism
  AUTO_POWER_OFF      = 2   // activate shut down
mechanism
};
```

System Time

```
typedef long SYSTIME;       // [ms]
```

14.3 FUNCTIONS

14.3.1 SUP_GetConfig - Get power management configuration status

C-Declaration

```
SUP_GetConfig(ON_OFF_TYPE &LowTempOnOff,
              SUP_AUTO_POWER &AutoPower,
              SYSTIME &Timeout)
```

VB-Declaration

```
VB_SUP_GetConfig(LowTempOnOff As Integer,
                 AutoPower As Integer,
                 Timeout As Long)
```

ASCII-Request

```
%R1Q,14001:
```

ASCII-Response

`%R1P,0,0:`*RC,LowTempOnOff[long], AutoPower[long], Timeout[long]*

Remarks

The returned settings are power off configuration and timing.

Parameters

`LowTempOnOff`	`out`	Current state of the low temperature control flag. If the `state=ON` the device automatically turns off when internal temperature fall short of -30°C.
`AutoPower`	`out`	Current activated shut down mechanism
`Timeout`	`out`	The timeout in ms. After this time the device switches in the mode defined by the value of `AutoPower` when no user activity (press a key, turn the device or communication via GeoCOM) occurs.

Return-Codes

`RC_OK`	Execution always successful.

See Also

```
SUP_SetConfig
SUP_SwitchLowTempControl
```

Example

see SUP_SetConfig

14.3.2 SUP_SetConfig - Set power management configuration

C-Declaration

```
SUP_SetConfig(ON_OFF_TYPE LowTempOnOff,
              SUP_AUTO_POWER AutoPower,
              SYSTIME Timeout)
```

VB-Declaration

```
VB_SUP_SetConfig(LowTempOnOff As Integer,
                 AutoPower As Integer,
                 Timeout As Long)
```

ASCII-Request

%R1Q,14002:*LowTempOnOff[long], AutoPower[long], Timeout[long]*

ASCII-Response

%R1P,0,0:*RC*

Remarks

Set the configuration for the low temperature control (ON|OFF), the auto power off automatic (AUTO_POWER_DISABLED|..._SLEEP|..._OFF) and the corresponding timeout for the auto power off automatic.

Parameters

LowTempOnOff	in	Switch for the low temperature control. Per default the device automatically turns off when internal temperature fall short of -30°C (LowTempOnOff = On).
AutoPower	in	Defines the behaviour of the power off automatic (default: AutoPower = AUTO_POWER_SLEEP).
Timeout	in	The timeout in ms. After this time the device switches in the mode defined by the value of AutoPower when no user activity (press a key, turn the device or communication via GeoCOM) occurs. The default value for Timeout is 900000ms = 15 Min.

Return-Codes

RC_OK	Execution always successful.

See Also

SUP_GetConfig
SUP_SwitchLowTempControl

Example

```
RC_TYPE                  rc;
ON_OFF_TYPE              LowTempOnOff;
SUP_AUTO_POWER           AutoPower;
SYSTIME                  Timeout;

// get parameter values
rc = SUP_GetConfig (LowTempOnOff,
                    AutoPower,
                    Timeout);


// set new values for parameter
LowTempOnOff   = OFF;
AutoPower      = AUTO_POWER_DISABLED;
Timeout        = 600000;  // =10min

rc = SUP_SetConfig (LowTempOnOff,
                    AutoPower,
                    Timeout);
```

14.3.3 SUP_SwitchLowTempControl - Set low temperature control

C-Declaration

```
SUP_SwitchLowTempControl(ON_OFF_TYPE LowTempOnOff)
```

VB-Declaration

```
VB_SUP_SwitchLowTempControl(LowTempOnOff As Integer)
```

ASCII-Request

%R1Q,14003:*LowTempOnOff[long]*

ASCII-Response

%R1P,0,0:*RC*

Remarks

Activate (ON) respectively deactivate (OFF) the low temperature control.

Parameters

`LowTempOnOff`	`in`	Switch for the low temperature control. Per defaults the device automatically turns off when internal temperature fall short of -30°C.

Return-Codes

`RC_OK`	Execution always successful.

See Also

`SUP_GetConfig`
`SUP_SetConfig`

Example

```
RC_TYPE          rc;

// deactivate the low temperature control

rc = SUP_SwitchLowTempControl(OFF)
```

15 THEODOLITE MEASUREMENT AND CALCULATION - TMC

This module is the central measurement, calculation and geodetic control module of the TPS1000 instrument family. All sensors (angle, distance and compensator) deliver their respective data to this module. All sensor information is used to continuously calculate corrected or uncorrected values for angles, distance and position co-ordinates.

The functions handled by the TMC module are:

Measurement Functions

These functions deliver measurement results. Angle- and inclination measurement are started by system functions directly, other measurement operations needs activating the corresponding sensor (e.g. distance measurement)

Measurement Control Functions

These functions control measurement behaviour (activate/deactivate sensors) and basic data for the calculation of measurement results.

Data Set-up Functions

These functions allow sending destination data, location data and section data to the Theodolite.

Information Functions

These functions return additional information about measurement results, sensors, Theodolite states, etc.

Configuration Functions

These functions control the Theodolite behaviour in general.

The measurement functions of this subsystem generally can generate three types of return codes:

System Return Codes are of general use (`RC_OK` means result is okay,...)

Informative Return code indicates that the function was terminated successfully. But some restrictions apply (e.g. it can be reported that the angle values are okay, the distance is invalid).

Error Return Codes signal a non-successful termination of the function call.

15.1 USAGE

15.1.1 Inclination measurement/correction

The TMC module handles the inclination sensor data and correction. To get exact results (co-ordinates, angles, distances) the inclination of the instrument must be taken into account. In general, there are two ways how this can be done:

Measuring the inclination

Calculating the inclination
For a limited time of several seconds and a limited horizontal angle between 10 and 40 degrees (depending on instrument type) an inclination model is generated to speed up measurement. The model for the inclination is based on the last exact inclination measurement and is maintained within the TMC as a calculated inclination plane.

To control the kind of generating the results, all measurement functions have a parameter (of type `TMC_INCLINE_PRG`), where the inclination mode can be selected. The different measurement modes are:

`TMC_MEA_INC`:
Measures the inclination (in any case). Use this mode by unstable conditions like e.g. the instrument has been moved or walking around the instrument may influence on an unstable underground (e.g. field grass). The disadvantage of this mode is that it is about half a second slower than `TMC_PLANE_INC`.

`TMC_PLANE_INC`:
Calculates the inclination (assumes that the instrument has not been moved). This mode gives an almost immediate result (some milliseconds).

`TMC_AUTO_INC`:
The system decides which method should be used (either `TMC_MEA_INC` or `TMC_PLANE_INC`). You get the best performance regarding measure rate and accuracy with this mode, the instrument checks the conditions around the station. We recommend taking this mode any time.

Note that the results depend on the system's configuration, too. That means that the compensator must be switched on in order to get a result with inclination correction (see `TMC_SetInclineSwitch`). The return code of the measurement functions holds information about the quality of the result. E.g. it is reported, if the compensation of inclination could not be done.

15.1.2 Sensor measurement programs

The instrument supports different measurement programs, which activates or deactivates the sensors in different manner. The programs can be selected by the control function `TMC_DoMeasure` (via the parameter of the type `TMC_MEASURE_PRG`).

Additionally the setting of the EDM measurement mode is set with the function `TMC_SetEdmMode` and influences the measurement. Here is a choice between single measurement and continues measurement is possible (each is different in speed and precision).

General measurement programs:

`TMC_DEF_DIST`:
Starts the distance measurement with the set distance measurement program.

`TMC_TRK_DIST`:
Starts the distance measurement in tracking mode.

`TMC_RTRK_DIST`:
Starts the distance measurement in rapid tracking mode.

`TMC_STOP`:
Stops measurement.

`TMC_CLEAR`:
Stops the measurement and clears the data.

`TMC_SIGNAL`:
Help mode for signal intensity measurement (use together with function `TMC_GetSignal`)

15.2 CONSTANTS AND TYPES

Inclination Sensor Measurement Program

(see Chapter Inclination measurement/correction for further information!)

```
enum TMC_INCLINE_PRG {
  TMC_MEA_INC    = 0,   // Use sensor (apriori sigma)
  TMC_AUTO_INC   = 1,   // Automatic mode (sensor/plane)
  TMC_PLANE_INC = 2,   // Use plane
}
```

TMC Measurement Mode

(see Chapter Sensor measurement programs for further information!)

```
enum TMC_MEASURE_PRG
  TMC_STOP       = 0,   // Stop measurement program
  TMC_DEF_DIST   = 1,   // Default DIST-measurement program
  TMC_TRK_DIST   = 2,   // Distance-TRK measurement program
  TMC_CLEAR      = 3,   // TMC_STOP and clear data
  TMC_SIGNAL     = 4,   // Signal measurement (test function)
  TMC_RTRK_DIST = 8    // Distance-TRK measurement program
};
```

EDM Measurement Mode

(see Chapter Sensor measurement programs for further information!)

```
enum EDM_MODE {
  EDM_SINGLE_STANDARD = 0, // Standard single measurement
  EDM_SINGLE_EXACT    = 1, // Exact single measurement
  EDM_SINGLE_FAST     = 2, // Fast single measurement
  EDM_CONT_STANDARD   = 3, // Repeated measurement
  EDM_CONT_EXACT      = 4, // Repeated average measurement
  EDM_CONT_FAST       = 5, // Fast repeated measurement
  EDM_UNDEFINED       = 6  // Not defined
};
```

Calculated Co-ordinates based on a Distance Measurement

```
struct TMC_COORDINATE {
  double dE;               // E-Coordinate
  double dN;               // N-Coordinate
  double dH;               // H-Coordinate
  SYSTIME CoordTime;       // Moment of dist. measurement
  double dE_Cont;          // E-Coordinate (continuously)
  double dN_Cont;          // N-Coordinate (continuously)
  double dH_Cont;          // H-Coordinate (continuously)
  SYSTIME CoordContTime;   // Moment of measurement [ms]
};
```

Corrected Angle Data

```
struct TMC_HZ_V_ANG {
  double dHz;               // Horizontal angle [rad]
  double dV;                // Vertical angle [rad]
}
```

Corrected Angle Data with Inclination Data

```
struct TMC_ANGLE {
  double dHz;              // Horizontal angle [rad]
  double dV;               // Vertical angle [rad]
  double dAngleAccuracy;   // Accuracy of angles [rad]
  SYSTIME AngleTime;       // Moment of measurement [ms]
  TMC_INCLINE Incline;     // Corresponding  inclination
  TMC_FACE eFace;          // Face position of telescope
};
```

Inclination Data

```
struct TMC_INCLINE {
  double dCrossIncline;    // Transverse axis incl. [rad]
  double dLengthIncline;   // Longitud. axis inclination [rad]
  double dAccuracyIncline; // Inclination accuracy [rad]
  SYSTIME InclineTime;     // Moment of measurement [ms]
};
```

System Time

```
typedef long SYSTIME;        // time since poweron [ms]
```

Face Position

```
enum TMC_FACE_DEF {
  TMC_FACE_NORMAL,         // Face in normal position
  TMC_FACE_TURN            // Face turned
};
```

Reflector Height

```
struct TMC_HEIGHT {
  double dHr;               // Reflector height
};
```

Atmospheric Correction Data

```
struct TMC_ATMOS_TEMPERATURE {
  double  dLambda;          // Wave length of the EDM transmitter
  double  dPressure;        // Atmospheric pressure
  double  dDryTemperature;  // Dry temperature
```

```
  double  dWetTemperature;  // Wet temperature
};
```

Refraction Control Data

```
struct TMC_REFRACTION {
  ON_OFF_TYPE eRefOn          // Refraction correction On/Off
  double  dEarthRadius;       // Radius of the earth
  double  dRefractiveScale;   // Refractive coefficient
};
```

Instrument Station Co-ordinates

```
struct TMC_STATION {
  double dE0;               // Station easting coordinate
  double dN0;               // Station northing coordinate
  double dH0;               // Station height coordinate
  double dHi;               // Instrument height
};
```

EDM Signal Information

```
struct TMC_EDM_SIGNAL  {
  double  dSignalIntensity; // Signal intensity of EDM in %
  SYSTIME Time;             // Time when measurement was taken
};
```

Correction Switches

```
struct TMC_ANG_SWITCH {
  ON_OFF_TYPE eInclineCorr;        // Inclination correction
  ON_OFF_TYPE eStandAxisCorr;      // Standing axis corr.
  ON_OFF_TYPE eCollimationCorr;    // Collimation error corr.
  ON_OFF_TYPE eTiltAxisCorr;       // Tilting axis corr.
};
```

15.3 MEASUREMENT FUNCTIONS

15.3.1 TMC_GetCoordinate – Gets the coordinates of a measured point

C-Declaration

```
TMC_GetCoordinate(SYSTIME WaitTime,
                  TMC_COORDINATE &Coordinate,
                  TMC_INCLINE_PRG Mode)
```

VB-Declaration

```
VB_TMC_GetCoordinate1(ByVal WaitTime As Long,
                      Coordinate As TMC_COORDINATE,
                      ByVal Mode As Integer)
```

ASCII-Request

`%R1Q,2082:`*WaitTime[long],Mode[long]*

ASCII-Response

`%R1P,0,0:`*RC,E[double],N[double],H[double],CoordTime[long], E-Cont[double],N-Cont[double],H-Cont[double],CoordContTime[long]*

Remarks

This function issues an angle measurement and, in dependence of the selected `Mode`, an inclination measurement and calculates the co-ordinates of the measured point with an already measured distance. The `WaitTime` is a delay to wait for the distance measurement to finish. Single and tracking measurements are supported. Information about a missing distance measurement and other information about the quality of the result is returned in the return- code.

Parameters

`WaitTime`	`in`	The delay to wait for the distance measurement to finish [ms].
`Coordinate`	`out`	Calculated Cartesian co-ordinates.
`Mode`	`in`	Inclination sensor measurement mode

Return Codes

`RC_OK`	Execution successful.

TMC_ACCURACY_GUARANTEE	Accuracy is not guaranteed, because the result are consist of measuring data which accuracy could not be verified by the system. Co-ordinates are available.
TMC_NO_FULL_CORRECTION	The results are not corrected by all active sensors. Co-ordinates are available. In order to secure witch correction is missing use the both functions TMC_IfDataAzeCorrError and TMC_IfDataIncCorrError
TMC_ANGLE_OK	Angle values okay, but no valid distance. Co-ordinates are not available.
TMC_ANGLE_ACCURACY_ GUARANTEE	No distance data available but angle data are valid. The return code is equivalent to the TMC_ACCURACY_GUARANTEE and relates to the angle data. Co-ordinates are not available.
TMC_ANGLE_NO_FULL_ CORRECTION	No distance data available but angle data are valid. The return code is equivalent to the TMC_NO_FULL_CORRECTION and relates to the angle data. Co-ordinates are not available. Perform a distance measurement first before you call this function.
TMC_DIST_ERROR	No measuring, because of missing target point, co-ordinates are not available. Aim target point and try it again
TMC_DIST_PPM	No distance measurement respectively no distance data because of wrong EDM settings. The co-ordinates are not available. Set EDM –ppm and -mm to 0.
TMC_ANGLE_ERROR	Problems with angle res. incline sensor. A valid angle could not be measured. At repeated occur call service.

TMC_BUSY	TMC resource is locked respectively TMC task is busy. Repeat measurement.
RC_ABORT	Measurement through customer aborted.
RC_SHUT_DOWN	System power off through customer.

See Also

TMC_DoMeasure
TMC_IfDataAzeCorrError
TMC_IfDataIncCorrError

Example

```
RC_TYPE                    Result;
TMC_COORDINATE             Coordinate;

// make a single distance measurement first
Result=TMC_DoMeasure(TMC_DEF_DIST, TMC_AUTO_INC);

if(Result==RC_OK)
{// before you get the coordinates
     Result=TMC_GetCoordinate(1000,Coordinate,
                              TMC_AUTO_INC);
}

switch(Result)
{// result interpretation
case RC_OK:
     break;
     .
     .
// error handling
case ...:
     .
     .
default:
     break;
}
```

15.3.2 TMC_GetSimpleMea - Returns angle and distance measurement

C-Declaration

```
TMC_GetSimpleMea(SYSTIME WaitTime,
                 TMC_HZ_V_ANG &OnlyAngle,
                 double &SlopeDistance,
                 TMC_INCLINE_PRG Mode)
```

VB-Declaration

```
VB_TMC_GetSimpleMea(ByVal WaitTime As Long,
                    OnlyAngle As TMC_HZ_V_ANG,
                    SlopeDistance As Double,
                    ByVal Mode As Integer)
```

ASCII-Request

`%R1Q,2108:`*WaitTime[long],Mode[long]*

ASCII-Response

`%R1P,0,0:`*RC,Hz[double],V[double],SlopeDistance[double]*

Remarks

This function returns the angles and distance measurement data. The distance measurement will be set invalid afterwards. It is important to note that this command does not issue a new distance measurement.

If a distance measurement is valid the function ignores `WaitTime` and returns the results.

If no valid distance measurement is available and the distance measurement unit is not activated (by `TMC_DoMeasure` before the `TMC_GetSimpleMea` call) the `WaitTime` is also ignored and the angle measurement result is returned. So this function can be used instead of `TMC_GetAngle5`.

Information about distance measurement is returned in the return- code.

Parameters

`WaitTime`	`in`	The delay to wait for the distance measurement to finish [ms].
`OnlyAngle`	`out`	Result of the angle measurement.
`SlopeDistance`	`out`	Result of the distance measurement [m].
`Mode`	`in`	Inclination sensor measurement mode.

Return Codes

`RC_OK`	Execution successful.

TMC_NO_FULL_CORRECTION	The results are not corrected by all active sensors. Angle and distance data are available. In order to secure witch correction is missing use the both functions TMC_IfDataAzeCorrError and TMC_IfDataIncCorrError This message is to be considers as warning.
TMC_ACCURACY_GUARANTEE	Accuracy is not guaranteed, because the result consisting of measuring data which accuracy could not be verified by the system. Angle and distance data are available. You can a forced incline measurement perform or switch off the incline. This message is to be considers as info.
TMC_ANGLE_OK	Angle values okay, but no valid distance. Perform a distance measurement.
TMC_ANGLE_NO_FULL_CORRECTION	No distance data available but angle data are valid. The return code is equivalent to the TMC_NO_FULL_CORRECTION and relates to the angle data. Perform a distance measurement first before you call this function.
TMC_ANGLE_ACCURACY_GUARANTEE	No distance data available but angle data are valid. The return code is equivalent to the TMC_ACCURACY_GUARANTEE and relates to the angle data.
TMC_DIST_ERROR	No measuring, because of missing target point, angle data are available but distance data are not available. Aims target point and try it again.

TMC_DIST_PPM	No distance measurement respectively no distance data because of wrong EDM settings. Angle data are available but distance data are not available.
	Set EDM –ppm and -mm to 0.
TMC_ANGLE_ERROR	Problems with angle res. incline sensor. A valid angle could not be measured. Distance and angle data are not available.
	At repeated occur call service.
TMC_BUSY	TMC resource is locked respectively TMC task is busy. Distance and angle data are not available.
	Repeat measurement.
RC_ABORT	Measurement through customer aborted.
RC_SHUT_DOWN	System power off through customer.

See Also

TMC_DoMeasure
TMC_GetAngle5

Example

```
RC_TYPE         rc;
TMC_HZ_V_ANG    OnlyAngle;
double          SlopeDistance;

// activate distance measurement
rc = TMC_DoMeasure(TMC_DEF_DIST, TMC_AUTO_INC);
if (rc == RC_OK)
{
  // distance measurement successful
  rc = TMC_GetSimpleMea(3000,
                        OnlyAngle,
                        SlopeDistance,
                        TMC_MEA_INC);
  if (rc == RC_OK)
  {
    // use distance and angle values
  else
  {
    // something with TMC_GetSimpleMea went wrong
  }
else
```

```
{
  // something with dist. measurement went wrong
}
```

15.3.3 TMC_GetAngle1 - Returns complete angle measurement

C-Declaration

```
TMC_GetAngle(TMC_ANGLE &Angle,
             TMC_INCLINE_PRG Mode)
```

VB-Declaration

```
VB_TMC_GetAngle1(Angle As TMC_ANGLE,
                 ByVal Mode As Integer)
```

ASCII-Request

`%R1Q,2003:`*Mode[long]*

ASCII-Response

`%R1P,0,0:`*RC,Hz[double],V[double],AngleAccuracy[double], AngleTime[long],CrossIncline[double],LengthIncline[double], AccuracyIncline[double],InclineTime[long],FaceDef[long]*

Remarks

This function carries out an angle measurement and, in dependence of configuration, inclination measurement and returns the results. As shown the result is very comprehensive. For simple angle measurements use `TMC_GetAngle5` or `TMC_GetSimpleMea` instead.

Information about measurement is returned in the return code.

Parameters

`Angle`	`out`	Result of the angle measurement.
`Mode`	`in`	Inclination sensor measurement mode.

Return Codes

`RC_OK`	Execution successful.
`TMC_NO_FULL_CORRECTION`	The results are not corrected by all active sensors. Angle data are available. In order to secure witch correction is missing use the both functions `TMC_IfDataAzeCorrError` and `TMC_IfDataIncCorrError` This message is to be considers as warning.

TMC_ACCURACY_GUARANTEE	Accuracy is not guaranteed, because the result consisting of measuring data which accuracy could not be verified by the system. Angle data are available. You can a forced incline measurement perform or switch off the incline. This message is to be considers as info.
TMC_ANGLE_ERROR	Problems with angle res. incline sensor. A valid angle could not be measured. Angle data are not available. At repeated occur call service.
TMC_BUSY	TMC resource is locked respectively TMC task is busy. Angle data are not available. Repeat measurement.
RC_ABORT	Measurement through customer aborted.
RC_SHUT_DOWN	System power off through customer.

See Also

TMC_DoMeasure
TMC_GetAngle5
TMC_GetSimpleMea

Example

see TMC_GetAngle5

15.3.4 TMC_GetAngle5 - Returns simple angle measurement

C-Declaration

```
TMC_GetAngle(TMC_HZ_V_ANG &OnlyAngle,
             TMC_INCLINE_PRG Mode)
```

VB-Declaration

```
VB_TMC_GetAngle5(OnlyAngle As TMC_HZ_V_ANG,
                 ByVal Mode As Integer)
```

ASCII-Request

%R1Q,2107:*Mode[long]*

ASCII-Response

`%R1P,0,0:`*RC,Hz[double],V[double]*

Remarks

This function carries out an angle measurement and returns the results. In contrast to the function `TMC_GetAngle1` this function returns only the values of the angle. For simple angle measurements use or `TMC_GetSimpleMea` instead.

Information about measurement is returned in the return code.

Parameters

`Angle`	`out`	Result of the angle measurement.
`Mode`	`in`	Inclination sensor measurement mode.

Return Codes

`RC_OK`	Execution successful.
`TMC_NO_FULL_CORRECTION`	The results are not corrected by all active sensors. Angle data are available. In order to secure witch correction is missing use the both functions `TMC_IfDataAzeCorrError` and `TMC_IfDataIncCorrError` This message is to be considers as warning.
`TMC_ACCURACY_GUARANTEE`	Accuracy is not guaranteed, because the result consisting of measuring data which accuracy could not be verified by the system. Angle data are available. You can a forced incline measurement perform or switch off the incline. This message is to be considers as info.
`TMC_ANGLE_ERROR`	Problems with angle res. incline sensor. A valid angle could not be measured. Angle data are not available. At repeated occur call service.
`TMC_BUSY`	TMC resource is locked respectively TMC task is busy. Angle data are not available. Repeat measurement.

RC_ABORT	Measurement through customer aborted.
RC_SHUT_DOWN	System power off through customer.

See Also

TMC_DoMeasure
TMC_GetAngle5
TMC_GetSimpleMea

Example

```
RC_TYPE         Result;
TMC_ANGLE       Angle;
BOOLE           bExit,
                bAzeCorrError,
                bIncCorrError;
short           nCnt;

nCnt=0;
do
{
bExit=TRUE;

// Gets the whole angle data
Result=TMC_GetAngle(Angle, TMC_AUTO_INC);

switch(Result)
{
case RC_OK:
     // Execution successful
     break;
case TMC_NO_FULL_CORRECTION:
     TMC_IfDataAzeCorrError(bAzeCorrError);
     TMC_IfDataIncCorrError(bIncCorrError);
     if(bAzeCorrError)
     {
     // coordinates are not corrected with the Aze-
     // deviation correction
     }
     if(bIncCorrError)
     {
     // coordinates are not corrected with the
     // incline correction
     }
     break;
case TMC_ACCURACY_GUARANTEE:
     // perform a forced incline measurement,
```

```
        // see example TMC_QuickDist
        break;

    case TMC_BUSY:
        // repeat measurement
        bExit=FALSE;
    case RC_ABORT:
    case RC_SHUT_DOWN:
    default:
        break;
    }// end switch

    nCnt++;
    }while(!bExit && nCnt<3);
```

15.3.5 TMC_QuickDist – Returns slope-distance and hz-,v-angle

C-Declaration

```
TMC_QuickDist(   TMC_HZ_V_ANG &OnlyAngle,
                 double       &dSlopeDistance)
```

VB-Declaration

```
VB_TMC_QuickDist(       OnlyAngle       As
                        TMC_HZ_V_ANG,
                        dSlopeDistance  As Double)
```

ASCII- Request

```
%R1Q,2117:
```

ASCII-Response

`%R1P,0,0:`*RC,dHz[double],dV[double],dSlopeDistance[double]*

Remarks

The function waits until a new distance is measured and then it returns the angle and the slope-distance, but no co-ordinates. Is no distance available, then it returns the angle values (hz, v) and the corresponding return-code.

At the call of this function, a distance measurement will be started with the rapid-tracking measuring program. If the EDM already active with the standard tracking measuring program, the measuring program will not changed to rapid tracking. Generally if the EDM is not active, then the rapid tracking measuring program will be started, otherwise the used measuring program will not be changed.

In order to abort the current measuring program use the function `TMC_DoMeasure`.

This function is very good suitable for target tracking, where high data transfers are required.

Note: Due to performance reasons the used inclination will be calculated (only if incline is activated), so the basic data for the incline calculation is exact, at least two forced incline measurements should be performed in between. The forced incline measurement is only necessary if the incline of the instrument because of measuring assembly has been changed.
Use the function `TMC_GetAngle(TMC_MEA_INC, Angle)` for the forced incline measurement. (For the forced incline measurement, the instrument must be in stable state for more than 3sec.).

Parameters

`OnlyAngle`	`out`	measured Hz- and V- angle
`dSlopeDistance`	`out`	measured slope-distance

Return-Codes

`RC_OK`	Measurement ok. Angle and distance data are available.
`TMC_NO_FULL_CORRECTION`	The results are not corrected by all active sensors. Angle and distance data are available. In order to secure witch correction is missing use the both functions `TMC_IfDataAzeCorrError` and `TMC_IfDataIncCorrError` This message is to be considers as warning.
`TMC_ACCURACY_GUARANTEE`	Accuracy is not guaranteed, because the result consisting of measuring data which accuracy could not be verified by the system. Angle and distance data are available. You can perform a forced incline measurement or switch off the incline. This message is to be considers as info.
`TMC_ANGLE_OK`	Angle measuring data are valid, but no distance data available. (Possible reasons are: –time out period to short –target out of view) This message is to be considers as warning.

TMC_ANGLE_NO_ FULL_CORRECTION	Angle measuring data are valid, but not corrected by all active sensors. The distance data are not available. (Possible reasons are: -see return code TMC_ANGLE_OK) This message is to be considers as warning.
TMC_ANGLE_ ACCURACY_ GUARANTEE	Angle measuring data are valid, but the accuracy is not guarantee, because the result (angle) consisting of measuring data, which accuracy could not be verified by the system. The distance data are not available. (Possible reasons are: -see return code TMC_ANGLE_OK) This message is to be considers as info.
TMC_DIST_ERROR	Because of missing target point no distance data available, but the angle data are valid respectively available. Aim target point and try it again.
TMC_DIST_PPM	No distance measurement respectively no distance data because of wrong EDM settings. The angle data are valid. Set EDM –ppm and –mm to 0.
TMC_ANGLE_ERROR	Problems with angle res. incline sensor. At repeated occur call service.
TMC_BUSY	TMC resource is locked respectively TMC task is busy. Repeat measurement.
RC_ABORT	Measurement through customer aborted.
RC_SHUT_DOWN	System power off through customer.

See Also

```
TMC_GetAngle
TMC_DoMeasure
TMC_IfDataAzeCorrError
TMC_IfDataIncCorrError
```

Example

```
const short     MAX=100;// number of measurements
const double    STATIC_TIME=4.0;// in seconds
```

```
const double    MAX_DIFFERENCE=0.0002// in rad
RC_TYPE         Result;
TMC_ANG_SWITCH  SwCorr;
TMC_HZ_V_ANG    HzVAng;
TMC_ANGLE       AngleDummy;
BOOLE           bExit;
DATIME          Datime;
double          dSlopeDist,
                dLastHzAng,
                dhz_angle_diff,
                dact_time, dstart_time;
short           nNoMeasurements;


TMC_GetAngSwitch(SwCorr);

SwCorr.eInclineCorr=ON;    // measure rate will be
SwCorr.eStandAxisCorr=ON;  // reduced if angle and
SwCorr.eCollimationCorr=ON;// incline correction are
SwCorr.eTiltAxisCorr=ON;   // activated

TMC_DoMeasure(TMC_CLEAR);  // clear distance first
TMC_SetAngSwitch(SwCorr);  // before you can set the
                           // ANG switches, the
                           // distance must be
                           // cleared

CSV_GetDateTime(Datime);
dstart_time=Datime.Time.Minute*60+
             Datime.Time.Second;

// starts the rapid tracking dist. measurement program
TMC_QuickDist(HzVAng, dSlopeDist);

bExit=FALSE;
nNoMeasurements=0;
do
{
     dLastHzAng=HzVAng.dHz;
     Result=TMC_QuickDist(HzVAng, dSlopeDist);
     switch(Result)
     {
     // distance- and angles- data available
     case TMC_ACCURACY_GUARANTEE:
          // perform a forced incline measurement
```

```
            // caution: the calculation at zero rad is
            // not consider
            dhz_angle_diff=fabs(dLastHzAng-
                                HzVAng.dHz);

            if(dhz_angle_diff<MAX_DIFFERENCE)
            {// instrument is in static period
                 CSV_GetDateTime(Datime);
                 dact_time=Datime.Time.Minute*60+
                           Datime.Time.Second;

                 if(dact_time-dstart_time > STATIC_TIME)
                 {// static mode exceeding 3-4 sec
                           TMC_GetAngle(TMC_MEA_INC,
                                        AngleDummy);
                           TMC_GetAngle(TMC_MEA_INC,
                                        AngleDummy);
                 }
            }
            else
            {// instrument is not in static period
                 CSV_GetDateTime(Datime);
                 dstart_time=Datime.Time.Minute*60+
                             Datime.Time.Second;
            }

       case RC_OK:
       case TMC_NO_FULL_CORRECTION:
            break;

       // no distance data available
       case TMC_ANGLE_OK:
       case TMC_ANGLE_NO_FULL_CORRECTION:
       case TMC_ANGLE_ACCURACY_GUARANTEE:
       case TMC_DIST_ERROR:
       case TMC_DIST_PPM:
            break;

       // neither angle- nor distance- data available
       case TMC_ANGLE_ERROR:
       case RC_BUSY:
       case RC_ABORT:
       case RC_SHUT_DOWN:
```

```
        default:
                bExit=TRUE;
                break;
        }
}
while(!bExit && nNoMeasurements<MAX);

TMC_DoMeasure(TMC_STOP);// stop measureprogram
```

15.4 MEASUREMENT CONTROL FUNCTIONS

15.4.1 TMC_DoMeasure - Carries out a distance measurement

C-Declaration

```
TMC_DoMeasure(TMC_MEASURE_PRG Command,
              TMC_INCLINE_PRG Mode)
```

VB-Declaration

```
VB_TMC_DoMeasure(ByVal Command As Integer,
                 ByVal Mode As Integer)
```

ASCII-Request

`%R1Q,2008:`*Command[long],Mode[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function carries out a distance measurement in a variety of TMC measurement modes like single distance, rapid tracking,... . Please note that this command does not output any values (distances). In order to get the values you have to use other measurement functions such as `TMC_GetCoordinate`, `TMC_GetSimpleMea` or `TMC_GetAngle`.

The value of the distance measured is kept in the instrument up to the next `TMC_DoMeasure` command where a new distance is requested or the distance is clear by the measurement program `TMC_CLEAR`.

Note: If you perform a distance measurement with the measure program TMC_DEF_DIST, the distance sensor will be work with the set EDM mode, see TMC_SetEdmMode.

Parameters

Command	in	TMC measurement mode.
Mode	in	Inclination sensor measurement mode.

Return-Codes

RC_OK	Execution successful.

See Also

TMC_SetEdmMode
TMC_GetCoordinate
TMC_GetSimpleMea
TMC_GetAngle1
TMC_GetAngle5

Example

```
RC_TYPE Result;
short   nCnt;

// set average mode
Result=TMC_SetEdmMode(EDM_CONT_EXACT);
// perform a single distance measurement
Result=TMC_DoMeasure(TMC_DEF_DIST);

nCnt=0;
while(nCnt<100)
{// wait on the distance data max. 100x100ms
     Result=TMC_GetCoordinate(100,Coordinate,
                                        TMC_AUTO_INC);
     nCnt++;
}

// to complete the measurement, and clear data
TMC_DoMeasure(TMC_CLEAR);
// set standard mode
TMC_SetEdmMode(EMD_SINGLE_STANDARD);
```

15.4.2 TMC_SetHandDist - Input slope distance and height offset

C-Declaration

```
TMC_SetHandDist(double SlopeDistance,
                double HgtOffset,
                TMC_INCLINE_PRG Mode)
```

VB-Declaration

```
VB_TMC_SetHandDist(ByVal SlopeDistance As Double,
                   ByVal HgtOffset As Double,
                   ByVal Mode As Integer)
```

ASCII-Request

`%R1Q,2019:`*SlopeDistance[double],HgtOffset[double],Mode[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function is used to input manually measured slope distance and height offset for a following measurement. Additionally an inclination measurement and an angle measurement are carried out to determine the co-ordinates of target. The V-angle is corrected to $\pi/2$ or $3 \cdot \pi/2$ in dependence of the instrument's face because of the manual input.

After the function call the previous measured distance is cleared.

Parameters

`SlopeDistance`	`in`	Slope distance
`HgtOffset`	`in`	Height offset
`Mode`	`in`	Inclination sensor measurement mode.

Return Codes

`RC_OK`	Execution successful.
`TMC_NO_FULL_CORRECTION`	The results are not corrected by all active sensors. In order to secure witch correction is missing use the both functions `TMC_IfDataAzeCorrError` and `TMC_IfDataIncCorrError` This message is to be considers as warning.

TMC_ACCURACY_GUARANTEE	Accuracy is not guaranteed, because the result consisting of measuring data which accuracy could not be verified by the system
	You can a forced incline measurement perform or switch off the incline.
	This message is to be considers as info.
TMC_ANGLE_ERROR	Problems with angle res. incline sensor. A valid angle could not be measured.
	At repeated occur call service.
TMC_BUSY	TMC resource is locked respectively TMC task is busy.
	Repeat measurement.
RC_ABORT	Measurement through customer aborted.
RC_SHUT_DOWN	System power off through customer.
RC_IVPARAM	Invalid parameter

See Also

TMC_IfDataAzeCorrError
TMC_IfDataIncCorrError

Example

```
RC_TYPE                  rc;
TMC_COORDINATE           Coordinate

rc = VB_TMC_SetHandDist(10, 1, TMC_AUTO_INC)
if (rc == RC_OK)
{
  // calculate coordinates
  rc=TMC_GetCoordinate(1000,Coordinate,TMC_AUTO_INC)
  if (rc == RC_OK)
  {
    // use coordinates
  else
  {
    // something went wrong
  }
}
```

15.5 DATA SETUP FUNCTIONS

15.5.1 TMC_GetHeight - Returns the current reflector height

C-Declaration

```
TMC_GetHeight(TMC_HEIGHT &Height)
```

VB-Declaration

```
VB_TMC_GetHeight(Height As TMC_HEIGHT)
```

ASCII-Request

```
%R1Q,2011:
```

ASCII-Response

```
%R1P,0,0:RC,Height[double]
```

Remarks

This function returns the current reflector height.

Parameters

Height	out	current reflector height

Return Codes

RC_OK	Execution always successful.

See Also

```
TMC_SetHeight
```

Example

```
RC_TYPE         rc;
TMC_HEIGHT      Height, NewHeight;

// reset reflector height to 0
// if it is not already

rc = TMC_GetHeight(Height);
if (Height.dHr != 0)
{
  NewHeight.dHr = 0;
  rc = TMC_SetHeight(NewHeight);
  if (rc == RC_OK)
  {
    // set of height successful
  }
  else
```

```
  {
    // TMC is busy, no set possible
  }
}
```

15.5.2 TMC_SetHeight - Sets new reflector height

C-Declaration

```
TMC_SetHeight(TMC_HEIGHT Height)
```

VB-Declaration

```
VB_TMC_SetHeight(ByVal Height As TMC_HEIGHT)
```

ASCII-Request

`%R1Q,2012:`*Height[double]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function sets a new reflector height.

Parameters

`Height`	`in`	new reflector height

Return Codes

`RC_OK`	Execution successful (new height is set).
`TMC_BUSY`	TMC resource is locked respectively TMC task is busy. The reflector height is not set. Repeat measurement.

See Also

`TMC_GetHeight`

Example

see `TMC_GetHeight`

15.5.3 TMC_GetAtmCorr - Get atmospheric correction parameters

C-Declaration

```
TMC_GetAtmCorr
       (TMC_ATMOS_TEMPERATURE &AtmTemperature)
```

VB-Declaration

```
VB_TMC_GetAtmCorr
        (AtmTemperature As TMC_ATMOS_TEMPERATURE)
```

ASCII-Request

```
%R1Q,2029:
```

ASCII-Response

`%R1P,0,0:`*RC,Lambda[double],Pressure[double],*
DryTemperature[double],WetTemperature[double]

Remarks

This function is used to get the parameters for the atmospheric correction.

Parameters

`AtmTemperature`	`out`	Atmospheric Correction Data

Return Codes

`RC_OK`	Execution always successful.

See Also

`TMC_SetAtmCorr`

Example

`see TMC_SetAtmCorr`

15.5.4 TMC_SetAtmCorr - Set atmospheric correction parameters

C-Declaration

```
TMC_SetAtmCorr
        (TMC_ATMOS_TEMPERATURE AtmTemperature)
```

VB-Declaration

```
VB_TMC_SetAtmCorr
     (ByVal AtmTemperature As TMC_ATMOS_TEMPERATURE)
```

ASCII-Request

`%R1Q,2028:`*Lambda[double],Pressure[double],*
DryTemperature[double],WetTemperature[double]

ASCII-Response

`%R1P,0,0:`*RC,*

Remarks

This function is used to set the parameters for the atmospheric correction.

Parameters

AtmTemperature	in	Atmospheric Correction Data

Return Codes

RC_OK	Execution successful (new atmospheric correction data are set).

See Also

TMC_GetAtmCorr

Example

```
TMC_ATMOS_TEMPERATURE AtmCorr;

TMC_GetAtmCorr(AtmCorr);

// set new wet and dry temperature
AtmCorr.dDryTemperature=60;
AtmCorr.dWetTemperature=80;

TMC_SetAtmCorr(AtmCorr);
```

15.5.5 TMC_SetOrientation - Orients the theodolite in Hz direction

C-Declaration

```
TMC_SetOrientation(double HzOrientation)
```

VB-Declaration

```
VB_TMC_SetOrientation(ByVal HzOrientation As Double)
```

ASCII-Request

%R1Q,2113:*HzOrientation[double]*

ASCII-Response

%R1P,0,0:*RC*

Remarks

This function is used to orientates the instrument in Hz direction. It is a combination of an angle measurement to get the Hz offset and afterwards setting the angle Hz offset in order to orientates onto a target. Before the new orientation can be set an existing distance must be cleared (use TMC_DoMeasure with the command = TMC_CLEAR).

Parameters

HzOrientation	in	Hz Orientation [rad]

Return Codes

RC_OK	Execution successful.
TMC_NO_FULL_CORRECTION	The orientation is set but not corrected by all active sensors. In order to secure witch correction is missing use the both functions TMC_IfDataAzeCorrError and TMC_IfDataIncCorrError This message is to be considers as warning.
TMC_ACCURACY_GUARANTEE	The orientation is set but the accuracy is not guarantee, because the result consisting of measuring data which accuracy could not be verified by the system You can a forced incline measurement perform or switch off the incline. This message is to be considers as info.
TMC_ANGLE_ERROR	Problems with angle res. incline sensor. A valid angle could not be measured. The orientation is not set. At repeated occur call service.
TMC_BUSY	TMC resource is locked respectively TMC task is busy or a distance is existing. The orientation is not set. Clear distance and repeat measurement.
RC_ABORT	Measurement through customer aborted.
RC_SHUT_DOWN	System power off through customer.

See Also

TMC_IfDataAzeCorrError
TMC_IfDataIncCorrError
TMC_DoMeasure

Example

```
RC_TYPE Result;

// clear existing distance first
TMC_DoMeasure(TMC_CLEAR);
```

```
// set orientation to 0
Result=TMC_SetOrientation(0.0);
if(Result!=RC_OK)
{
// error or warning handling
}
```

15.5.6 TMC_GetPrismCorr - Get the prism constant

C-Declaration

```
TMC_GetPrismCorr(double &PrismCorr)
```

VB-Declaration

```
VB_TMC_GetPrismCorr(PrismCorr As Double)
```

ASCII-Request

```
%R1Q,2023:
```

ASCII-Response

```
%R1P,0,0:RC,PrismCorr[double]
```

Remarks

This function is used to get the prism constant.

Parameters

PrismCorr	out	Prism constant [mm]

Return Codes

RC_OK	Execution always successful.

See Also

```
TMC_SetPrismCorr
```

Example

```
const double Corr = 0.1;
RC_TYPE   rc;
double    PrismCorr;

// set the prism constant to
// 0.1 if not already set

rc = TMC_GetPrismCorr(PrismCorr);
if (PrismCorr != Corr)
{
  rc = TMC_SetPrismCorr(Corr);
```

```
    if (rc == RC_OK)
    {
      // set of prisma corr successful
    }
    else
    {
      // Invalid parameter
    }
}
```

15.5.7 TMC_SetPrismCorr - Set the prism constant

C-Declaration

`TMC_SetPrismCorr(double PrismCorr)`

VB-Declaration

`VB_TMC_SetPrismCorr(ByVal PrismCorr As Double)`

ASCII-Request

`%R1Q,2024:`*PrismCorr[double]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function is used to set the prism constant.

Parameters

`PrismCorr`	`in`	Prism constant [mm]

Return Codes

`RC_OK`	Execution successful.
`TMC_BUSY`	TMC resource is locked respectively TMC task is busy. The prism constant is not set. Repeat measurement.

See Also

`TMC_GetPrismCorr`

Example

see `TMC_GetPrismCorr`

15.5.8 TMC_GetRefractiveCorr - Get the refraction factor

C-Declaration

```
TMC_GetRefractiveCorr(TMC_REFRACTION &Refractive)
```

VB-Declaration

```
VB_TMC_GetRefractiveCorr
                  (Refractive As TMC_REFRACTION)
```

ASCII-Request

```
%R1Q,2031:
```

ASCII-Response

```
%R1P,0,0:RC,RefOn[boolean],EarthRadius[double], RefractiveScale[double]
```

Remarks

This function is used to get the refraction distortion factor for correction of measured height difference.

Parameters

`Refractive`	`out`	Refraction distortion

Return Codes

`RC_OK`	Execution always successful.

See Also

`TMC_SetRefractiveCorr`

Example

```
const double              EarthRadius = 6378000;
RC_TYPE                   rc;
TMC_REFRACTION  Refractive;

// check the earth radius setting
// and reset if necessary
rc = TMC_GetRefractiveCorr(Refractive);
if (Refractive.dEarthRadius != EarthRadius)
{
  Refractive.dEarthRadius = EarthRadius;
  rc = TMC_SetRefractiveCorr(Refractive);
  if (rc == RC_OK)
  {
    // set of earth radius successful
  }
  else
  {
    // set not successful (subsystem busy)
```

```
  }
}
```

15.5.9 TMC_SetRefractiveCorr - Set the refraction factor

C-Declaration

```
TMC_SetRefractiveCorr(TMC_REFRACTION Refractive)
```

VB-Declaration

```
VB_TMC_SetRefractiveCorr
                (ByVal Refractive As TMC_REFRACTION)
```

ASCII-Request

`%R1Q,2030:` *RefOn[boolean],EarthRadius[double], RefractiveScale[double]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function is used to set the refraction distortion factor for correction of measured height difference.

Parameters

`Refractive`	`in`	Refraction distortion

Return Codes

`RC_OK`	Execution successful.
`TMC_BUSY`	TMC resource is locked respectively TMC task is busy. The refraction distortion factor is not set. Repeat measurement.

See Also

`TMC_GetRefractiveCorr`

Example

see `TMC_GetRefractiveCorr`

15.5.10 TMC_GetRefractiveMethod - Get the refraction model

C-Declaration

```
TMC_GetRefractiveMethod(unsigned short &Method)
```

VB-Declaration

```
VB_TMC_GetRefractiveMethod(Method As Integer)
```

ASCII-Request

```
%R1Q,2091:
```

ASCII-Response

`%R1P,0,0:`*RC,Method[unsigned short]*

Remarks

This function is used to get the current refraction model.

Parameters

`Method`	`out`	Refraction data: `Method` = 1 means method 1 (for Australia) `Method` = 2 means method 1 (for the rest of the world)

Return Codes

`RC_OK`	Execution always successful.

See Also

`TMC_SetRefractiveMethod`

Example

```
const unsigned short      RefractiveMethod = 1;
RC_TYPE                    rc;
unsigned short             Method;

// set the refractive methode to 1
// if it is not already

rc = TMC_GetRefractiveMethod(Method);
if (Method != RefractiveMethod)
{
  rc = TMC_SetRefractiveMethod(RefractiveMethod);
  if (rc == RC_OK)
  {
    // set of refractive methode successful
  }
  else
```

```
  {
    // set not successful (subsystem busy)
  }
}
```

15.5.11 TMC_SetRefractiveMethod - Set the refraction model

C-Declaration

`TMC_SetRefractiveMethod(unsigned short Method)`

VB-Declaration

`VB_TMC_SetRefractiveMethod(ByVal Method As Integer)`

ASCII-Request

`%R1Q,2090:`*Method[unsigned short]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function is used to set the refraction model.

Parameters

`Method`	`in`	Refraction data: `Method` = 1 means method 1 (for Australia) `Method` = 2 means method 1 (for the rest of the world)

Return Codes

`RC_OK`	Execution successful.
`TMC_BUSY`	TMC resource is locked respectively TMC task is busy. The refraction model is not set. Repeat measurement.

See Also

`TMC_GetRefractiveMethod`

Example

see `TMC_GetRefractiveMethod`

15.5.12 TMC_GetStation - Get the coordinates of the instrument station

C-Declaration

```
TMC_GetStation(TMC_STATION &Station)
```

VB-Declaration

```
VB_TMC_GetStation(Station As TMC_STATION)
```

ASCII-Request

```
%R1Q,2009:
```

ASCII-Response

%R1P,0,0:*RC,E0[double],N0[double],H0[double],Hi[double]*

Remarks

This function is used to get the co-ordinates of the instrument station.

Parameters

Station	out	Instrument station co-ordinates.

Return Codes

RC_OK	Execution always successful.

See Also

TMC_SetStation

Example

```
RC_TYPE         rc;
TMC_STATION Station, NullStation;
NullStation.dE0 = 0;
NullStation.dN0 = 0;
NullStation.dH0 = 0;
NullStation.dHi = 0;

// reset station coordinates to 0

rc = TMC_GetStation(Station);
if ((Station.dE0 != 0)||
    (Station.dN0 != 0)||
    (Station.dH0 != 0)||
    (Station.dHi != 0))
{
rc = TMC_GetStation(NullStation);
  if (rc == RC_OK)
  {
    // reset of station successful
  }
```

```
  else
  {
    // reset not successful (subsystem busy)
  }
}
```

15.5.13 TMC_SetStation - Set the coordinates of the instrument station

C-Declaration

`TMC_SetStation(TMC_STATION Station)`

VB-Declaration

`VB_TMC_SetStation(ByVal Station As TMC_STATION)`

ASCII-Request

`%R1Q,2010:`*E0[double],N0[double],H0[double],Hi[double]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function is used to set the co-ordinates of the instrument station.

Parameters

`Station`	`in`	Instrument station co-ordinates.

Return Codes

`RC_OK`	Execution successful.
`TMC_BUSY`	TMC resource is locked respectively TMC task is busy or a distance is existing. The instrument co-ordinates are not set. Clear distance and repeat measurement.

See Also

`TMC_GetStation`
`TMC_DoMeasure`

Example

see `TMC_GetStation`

15.6 INFORMATION FUNCTIONS

15.6.1 TMC_GetFace - Get face information of current telescope position

C-Declaration

```
TMC_GetFace(TMC_FACE &Face)
```

VB-Declaration

```
VB_TMC_GetFace(Face As Integer)
```

ASCII-Request

```
%R1Q,2026:
```

ASCII-Response

%R1P,0,0:*RC,Face[long]*

Remarks

This function returns the face information of the current telescope position. The face information is only valid, if the instrument is in an active measurement state (that means a measurement function was called before the `TMC_GetFace` call, see example). Note that the instrument automatically turns into an inactive measurement state after a predefined timeout.

Parameters

`Face`	`out`	Face position.

Return Codes

`RC_OK`	Execution always successful.

See Also

```
AUT_ChangeFace
```

Example

```
RC_TYPE         rc;
TMC_FACE   Face;

// turn the face if not in normal position

// set active measurement state
rc = TMC_DoMeasure(TMC_DEF_DIST, TMC_AUTO_INC);
rc = TMC_GetFace(Face);
```

```
if (Face == TMC_FACE_TURN)
{
  rc = AUT_ChangeFace(AUT_NORMAL,
                      AUT_POSITION,
                      FALSE);
  if (rc == RC_OK)
  {
    // face successfuly turned
  }
  else
  {
    // change face problem: see AUT_ChangeFace
  }
}
// clear distance
rc = TMC_DoMeasure(TMC_CLEAR, TMC_AUTO_INC);
```

15.6.2 TMC_GetSignal - Get information about EDM's signal amplitude

C-Declaration

`TMC_GetSignal(TMC_EDM_SIGNAL &Signal)`

VB-Declaration

`VB_TMC_GetSignal(Signal As TMC_EDM_SIGNAL)`

ASCII-Request

`%R1Q,2022:`

ASCII-Response

`%R1P,0,0:`*RC,SignalIntensity[double],Time[long]*

Remarks

This function returns information about the amplitude of the EDM signal. The function only can perform measuring if the signal measurement program is activated. Start the signal measurement program with `TMC_DoMeasure` where `Command = TMC_SIGNAL`. After the measurement the EDM must be switch off (use `TMC_DoMeasure` where `Command = TMC_CLEAR`).

Parameters

`Face`	`out`	Face position.

Return Codes

`RC_OK`	Execution successful.

TMC_SIGNAL_ERROR	Error within signal measurement. At repeated occur call service.
RC_IVPARAM	The signal measurement program is not activated. No signal measurement possible. Activate signal measurement.
RC_ABORT	Measurement through customer aborted.
RC_SHUT_DOWN	System power off through customer.

See Also

TMC_DoMeasure

Example

```
RC_TYPE Result;
TMC_SIGNAL Signal;

TMC_DoMeasure(TMC_SIGNAL);
do
{
     Result=TMC_GetSignal(Signal);
     if(Result==RC_OK)
     {
          .
          .
          .
     }
}while(Result==RC_OK);
```

15.7 CONFIGURATION FUNCTIONS

15.7.1 TMC_GetAngSwitch - Get angular correction's states

C-Declaration

```
TMC_GetAngSwitch(TMC_ANG_SWITCH &SwCorr)
```

VB-Declaration

```
VB_TMC_GetAngSwitch(SwCorr As TMC_ANG_SWITCH)
```

ASCII-Request

```
%R1Q,2014:
```

ASCII-Response

`%R1P,0,0:`*RC,InclineCorr[long],StandAxisCorr[long], CollimationCorr[long],TiltAxisCorr[long]*

Remarks

This function returns the angular correction's state.

Parameters

`SwCorr`	`out`	Angular correction's state.

Return Codes

`RC_OK`	Execution always successful.

See Also

`TMC_SetAngSwitch`

Example

```
RC_TYPE         rc;
TMC_ANG_SWITCH  SwCorr;

// get the switch state for the angular
// correction

rc = TMC_GetAngSwitch(SwCorr);
if (SwCorr.eTiltAxisCorr == ON)
{
  // Tilting axis correction turned On
}
else
{
  // Tilting axis correction turned Off
}
```

15.7.2 TMC_GetInclineSwitch - Get the dual axis compensator's state

C-Declaration

```
TMC_GetInclineSwitch(ON_OFF_TYPE  &SwCorr)
```

VB-Declaration

```
VB_TMC_GetInclineSwitch(SwCorr As Integer)
```

ASCII-Request

```
%R1Q,2007:
```

ASCII-Response

`%R1P,0,0:`*RC,SwCorr[long]*

Remarks

This function returns the current dual axis compensator's state.

Parameters

`SwCorr`	`out`	Dual axis compensator's state.

Return Codes

`RC_OK`	Execution always successful.

See Also

`TMC_SetInclineSwitch`

Example

```
RC_TYPE         rc;
ON_OFF_TYPE     SwCorr;

// clear distance first before you change the state
TMC_DoMeasure(TMC_CLEAR, TMC_AUTO,INC);

// deactivate the compensator
// if it is not already

rc = TMC_GetInclineSwitch(SwCorr);
if (SwCorr == ON)
{
  rc = TMC_SetInclineSwitch(OFF);
  if (rc == RC_OK)
  {
    // successfully deactvated
  }
  else
  {
    // set not successful (subsystem busy)
  }
}
```

15.7.3 TMC_SetInclineSwitch - Switch dual axis compensator on or off

C-Declaration

```
TMC_SetInclineSwitch(ON_OFF_TYPE  SwCorr)
```

VB-Declaration

```
VB_TMC_SetInclineSwitch(ByVal SwCorr As Integer)
```

ASCII-Request

`%R1Q,2006:`*SwCorr[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function switches the dual axis compensator on or off.

Parameters

`SwCorr`	`in`	Dual axis compensator's state.

Return Codes

`RC_OK`	Execution successful.
`TMC_BUSY`	TMC resource is locked respectively TMC task is busy or a distance is existing. The incline state is not changed. Clear distance and repeat measurement.

See Also

`TMC_GetInclineSwitch`

Example

`see TMC_GetInclineSwitch`

15.7.4 TMC_GetEdmMode - Get the EDM measurement mode

C-Declaration

```
TMC_GetEdmMode(EDM_MODE &Mode)
```

VB-Declaration

```
VB_TMC_GetEdmMode(Mode As Integer)
```

ASCII-Request

`%R1Q,2021:`

ASCII-Response

`%R1P,0,0:`*RC,Mode[long]*

Remarks

This function returns the EDM measurement mode.

Parameters

Mode	out	EDM measurement mode.

Return Codes

RC_OK	Execution always successful.

See Also

TMC_SetEdmMode

Example

```
RC_TYPE         rc;
EDM_MODE   Mode;

// set EDM mode to single standard
// if it is in any repeated mode

rc = TMC_GetEdmMode(Mode);
switch (Mode)
{
  case (EDM_CONT_STANDARD):
  case (EDM_CONT_EXACT):
  case (EDM_CONT_FAST):
    rc = TMC_SetEdmMode(EDM_SINGLE_STANDARD);
    if (rc == RC_OK)
    {
      // set to single mode successful
    }
    else
    {
      // set not successful (subsystem busy)
    }
}
```

15.7.5 TMC_SetEdmMode - Set EDM measurement modes

C-Declaration

```
TMC_SetEdmMode(EDM_MODE Mode)
```

VB-Declaration

```
VB_TMC_SetEdmMode(ByVal Mode As Integer)
```

ASCII-Request

%R1Q,2020:*Mode[long]*

ASCII-Response

`%R1P,0,0:`*RC*

Remarks

This function set the current measurement modes new. The measure function `TMC_DoMeasure(TMC_DEF_DIST)` will work with this configuration.

Parameters

`Mode`	`in`	EDM measurement mode.

Return Codes

`RC_OK`	Execution successful.
`TMC_BUSY`	TMC resource is locked respectively TMC task is busy. The EDM mode is not set. Repeat measurement.

See Also

`TMC_GetEdmMode`
`TMC_DoMeasure`

Example

see `TMC_GetEdmMode`

15.7.6 TMC_GetSimpleCoord – Get cartesian coordinates

C-Declaration

```
TMC_GetSimpleCoord( SYSTIME WaitTime,
                    double &dCoordE,
                    double& dCoordN,
                    double& dCoordH,
                    TMC_INCLINE_PRG eProg)
```

VB-Declaration

```
VB_TMC_GetSimpleCoord(  ByVal WaitTime As SYSTIME,
                        dCoordE As Double,
                        dCoordN As Double,
                        dCoordH As Double,
                        ByVal eProg As TMC_INCLINE_PRG)
```

ASCII-Request

`%R1Q,2116:`*WaitTime[long],eProg[long]*

ASCII-Response

%R1P,0,0:*RC,dCoordE[double], dCoordN[double], dCoordH[double]*

Remarks

This function get the cartesian co-ordinates if a valid distance existing The parameter WaitTime defined the max wait time in order to get a valid distance. If after the wait time not a valid distance existing, the function initialise the parameter for the co-ordinates (E,N,H) with 0 and returns a error. For the co-ordinate calculate will require incline results. With the parameter eProg you have the possibility the incline results either to calculate or to measure it anew explicitly. We recommend to use the third variant, let the system determined (see parameters).

Parameters

WaitTime	in	Max. wait time to get a valid distance [ms]
eProg	in	Incline measuring program TMC_MEA_INC: Incline will explicit measured, the execution time for the function will so drastic increased respectively the measure rate will be slow down. **Note**: If you need a high accuracy then use this program: TMC_PLANE_INC: incline will be calculated, this variant is the fastest way to get the co-ordinates. **Note**: If you need a high measure rate and the station got a high stability on tilting use this program. TMC_AUTO_INC: the system decides whether the incline must be calculated or measured. **Note**: The best performance regarding measure rate and accuracy you get with the automatically program, the instrument checks the conditions around the station. We recommend to take this mode any time.
dCoordE	out	eastern co-ordinate

dcoordN	out	northern co-ordinate
dcoordH	out	high co-ordinate

Return-Codes

RC_OK	Measurement ok
TMC_NO_FULL_CORRECTION	The results are not corrected by all active sensors. Co-ordinates are available. In order to secure witch correction is missing use the both functions TMC_IfDataAzeCorrError and TMC_IfDataIncCorrError
TMC_ACCURACY_GUARANTEE	Accuracy is not guaranteed, because the result are consist of measuring data which accuracy could not be verified by the system. Co-ordinates are available.
TMC_DIST_PPM	Wrong EDM settings, co-ordinates are not valid and set to 0. Set EDM ppm value to 0.
TMC_DIST_ERROR	No measuring, because of missing target point, co-ordinates are not valid and set to 0 Aim target point and try it again
TMC_BUSY	TMC resource is locked respectively TMC task is busy, co-ordinates are not valid and set to 0. Repeat measurement.
TMC_ANGLE_ERROR	Problems with angle res. incline sensor. At repeated occur call service.
RC_ABORT	Measurement through customer aborted.
RC_SHUT_DOWN	System power off through customer
RC_IVRESULT	No distance existing, co-ordinates are not valid and set to 0. Execute a distance measurement first.

See Also

TMC_GetCoordinate
TMC_IfDataAzeCorrError

```
TMC_IfDataIncCorrError
```

Example

```
RC_TYPE         Result;
TMC_ANG_SWITCH  SwCorr;
SYSTIME         WaitTime;
TMC_INCLINE_PRG ePrgm;
BOOLE           bExit;
Double          dCoordE,dCoordN,dCoordH;


TMC_GetAngSwitch(SwCorr);   // measure rate will
SwCorr.eInclineCorr=ON;     // be reduced with
SwCorr.eStandAxisCorr=ON;   // angle and incline
SwCorr.eCollimationCorr=ON; // corrections.
SwCorr.eTiltAxisCorr=ON;
TMC_DoMeasure(TMC_CLEAR); // clear distance first
TMC_SetAngSwitch(SwCorr); // before you can set the
                          // ANG switches, the
                          // distance must be
                          // cleared

TMC_DoMeasure(TMC_RTRK_DIST);// execute rapid
                             // tracking
                             // measurement

WaitTime=500;// set max. wait time 500 [ms]
eProg=TMC_AUTO_INC;// set automatically incline prgm
bExit=FALSE;
do
{
Result=TMC_GetSimpleCoord(WaitTime, dCoordE,
                          dCoordN, dCoordH,eProg);
switch(Result)
{
case RC_OK:
case TMC_NO_FULL_CORRECTION:
case TMC_ACCURACY_GUARANTEE:
     // in this cases are the coordinates
     // available
Break;
Default:
     bExit=TRUE;
     // in all other cases are the coordinates not
     // valid and set to 0
     // further errorhandling
```

```
Break;
}// end switch
}// end do while
while(!bExit);

TMC_DoMeasure(TMC_CLEAR); // complete measurement
                          // and clear data
```

15.7.7 TMC_IfDataAzeError – If ATR error occur

C-Declaration

```
TMC_IfDataAzeCorrError(BOOLE& bAtrCorrectionError)
```

VB-Declaration

```
VB_TMC_IfDataAzeCorrError
                (bAtrCorrectionError As Integer)
```

ASCII-Request

```
%R1Q,2114:
```

ASCII-Response

```
%R1P,0,0:RC,bAtrCorrectionError[long]
```

Remarks

If you get back the return code TMC_ANGLE_NO_FULL_CORRECTION or TMC_ NO_FULL_CORRECTION from a measurement function, so you can find out with this function, whether the returned data record from the measurement function a missing deviation correction of the ATR included or not.

Parameters

`bAtrCorrectionError`	Out	Flag, if ATR correction error occurred or not `FALSE`: no error occurred `TRUE`: last data record not corrected with the ATR-deviation

Return-Codes

`RC_OK`	Always

See Also

TMC_IfDataIncCorrError

Example

```
RC_TYPE         Result;
SYSTIME         WaitTime;
TMC_INCLINE_PRG ePrgm;
double          dCoordE,dCoordN,dCoordH;

TMC_DoMeasure(TMC_DEF_DIST);// execute single
                            // dist measurement

WaitTime=500;// set max. wait time 500 [ms]
eProg=TMC_AUTO_INC;// set automatically incline prgm

Result=TMC_GetSimpleCoord(WaitTime, dCoordE,
                          dCoordN, dCoordH,eProg);
switch(Result)
{
case TMC_NO_FULL_CORRECTION:
     TMC_IfDataAzeCorrError(bAzeCorrError);
     TMC_IfDataIncCorrError(bIncCorrError);
     if(bAzeCorrError)
     {
     // coordinates are not corrected with the Aze-
     // deviation correction
     }
     if(bIncCorrError)
     {
     // coordinates are not corrected with the //
incline correction
     }
case RC_OK:
case TMC_ACCURACY_GUARANTEE:
     // in this cases are the coordinates
     // available
break;
default:
     // in all other cases are the coordinates not
     // valid and set to 0
     // further errorhandling
break;
}// end switch

TMC_DoMeasure(TMC_CLEAR); // complete measurement
```

```
                                    // and clear data
```

15.7.8 TMC_IfDataIncError – If incline error occur

C-Declaration

```
TMC_IfDataIncCorrError(BOOLE& bIncCorrectionError)
```

VB-Declaration

```
VB_TMC_IfDataIncCorrError
                  (bIncCorrectionError As Integer)
```

ASCII-Request

```
%R1Q,2115:
```

ASCII-Response

```
%R1P,0,0:RC,bIncCorrectionError[long]
```

Remarks

If you get back the return code TMC_ANGLE_NO_FULL_CORRECTION or TMC_ NO_FULL_CORRECTION from a measurement function, so you can find out with this function, whether the returned data record from the measurement function a missing inclination correction of the incline sensor included or not. A error information can only occur if the incline sensor is active.

Parameters

`bIncCorrectionError`	out	Flag, if incline correction error occurred or not `FALSE`: no error occurred `TRUE`: last data record not corrected with the incline-correction

Return-Codes

`RC_OK`	always

See Also

`TMC_IfDataAzeCorrError`

Example

```
see example TMC_IfDataAzeCorrError
```

15.7.9 TMC_SetAngSwitch – Enable/disable angle corrections

C-Declaration

```
TMC_SetAngSwitch(TMC_ANG_SWITCH Switch)
```

VB-Declaration

```
VB_TMC_SetAngSwitch(ByVal Switch As TMC_ANG_SWITCH)
```

ASCII-Request

```
%R1Q,2016:
```

ASCII-Response

```
%R1P,0,0:RC
```

Remarks

With this function you can enable/disable follow angle measurement correction.

`incline:`	The incline will be considered in the angle measurement if enabled.
`stand axis:`	The stand axis will be considered in the angle measurement if enabled.
`collimation:`	The collimation will be considered in the angle measurement if enabled
`tilt axis:`	The tilt axis will be considered in the angle measurement if enabled. This correction will be ignored for instruments of 1100 class.

Note: You can set the various corrections only, if no distance is existing! (Use the function `TMC_DoMeasure(TMC_CLEAR,..)` in order to clear the distance)

Parameters

`Switch`	Angle measurement corrections

Return-Code

`RC_OK`	Corrections are set
`TMC_BUSY`	TMC resource is locked respectively TMC task is busy or a distance is existing. Clear distance and try it again.

See-Also

`TMC_DoMeasure`

`TMC_GetAngSwitch`

Example

`See example TMC_QuickDist`

16 WI - REGISTRATION – WIR

This chapter describes in which format measurements should be stored on PC-Card. We distinguish between the old GSI – 8 format and the newly established GSI – 16 format. In the former case the data part is 8 bytes and in the latter case 16 bytes.

16.1 CONSTANTS

Record Format Constants

```
typedef short WIR_RECFORMAT;
const WIR_RECFORMAT WIR_RECFORMAT_GSI = 0;
    // defines recording format is GSI (standard)
const WIR_RECFORMAT WIR_RECFORMAT_GSI16 = 1;
    // defines recording format is the new GSI-16
```

16.2 FUNCTIONS

16.2.1 WIR_GetRecFormat – Get Record Format

C-Declaration

```
WIR_GetRecFormat(WIR_RECFORMAT &RecFormat )
```

VB-Declaration

```
VB_WIR_GetRecFormat( RecFormat As Integer )
```

ASCII-Request

```
%R1Q,8011:
```

ASCII-Response

```
%R1P,0,0:RC, RecFormat[short]
```

Remarks

This function retrieves which recording format is in use.

Parameters

RecFormat	out	WIR_RECFORMAT_GSI or WIR_RECFORMAT_GSI16

Return Codes

RC_OK	On successful termination.

See Also

WIR_SetRecFormat

Example

see WIR_SetRecFormat

16.2.2 WIR_SetRecFormat – Set Record Format

C-Declaration

```
WIR_SetRecFormat( WIR_RECFORMAT RecFormat )
```

VB-Declaration

```
VB_WIR_SetRecFormat( RecFormat As Integer)
```

ASCII-Request

```
%R1Q,8012:RecFormat[short]
```

ASCII-Response

```
%R1P,0,0:RC
```

Remarks

This function sets which recording format should be used.

Parameters

RecFormat	in	WIR_RECFORMAT_GSI or WIR_RECFORMAT_GSI16

Return Codes

RC_OK	On successful termination.

See Also

WIR_GetRecFormat

Example

```
// if in old GSI format then switch to new format
RetCode = WIR_GetRecFormat(Format);
if (Format == WIR_RECFORMAT_GSI)
{
  // switch to GSI16 format
  RetCode = WIR_SetRecFormat(WIR_RECFORMAT_GSI16);
}
```

A RETURN CODES

The return codes described here are codes, which may be returned from RPC's and GeoCOM general functions (COMF). A successful completion will be denoted by RC_OK. Almost all of the return codes are error codes. Nevertheless, some of them have a more informational character. Therefore, refer also to the description of a specific function. In a special context the meaning of a return code might vary a little bit.

The list described here is organised in subsystem related categories. The `RetCodeName` describes the constant as it is defined for the TPS1000 series instruments. Additionally to find an error code by number they are given too.

A-1 GENERAL RETURN CODES

RetCodeName	Val	Description
RC_OK	0	Function successfully completed.
RC_UNDEFINED	1	Unknown error, result unspecified.
RC_IVPARAM	2	Invalid parameter detected. Result unspecified.
RC_IVRESULT	3	Invalid result.
RC_FATAL	4	Fatal error.
RC_NOT_IMPL	5	Not implemented yet.
RC_TIME_OUT	6	Function execution timed out. Result unspecified.
RC_SET_INCOMPL	7	Parameter setup for subsystem is incomplete.
RC_ABORT	8	Function execution has been aborted.
RC_NOMEMORY	9	Fatal error – not enough memory.
RC_NOTINIT	10	Fatal error – subsystem not initialised.
RC_SHUT_DOWN	12	Subsystem is down.
RC_SYSBUSY	13	System busy/already in use of another process. Cannot execute function.
RC_HWFAILURE	14	Fatal error – hardware failure.
RC_ABORT_APPL	15	Execution of application has been aborted (SHIFT-ESC).
RC_LOW_POWER	16	Operation aborted - insufficient power supply level.

A-2 ANG SUBSYSTEM

RetCodeName	Val	Description
ANG_ERROR	257	Angles and Inclinations not valid
ANG_INCL_ERROR	258	inclinations not valid
ANG_BAD_ACC	259	value accuracy not reached
ANG_BAD_ANGLE_ACC	260	angle-accuracy not reached
ANG_BAD_INCLIN_ACC	261	inclination accuracy not reached
ANG_WRITE_PROTECTED	266	no write access allowed
ANG_OUT_OF_RANGE	267	value out of range
ANG_IR_OCCURED	268	function aborted due to interrupt
ANG_HZ_MOVED	269	Hz moved during incline measurement
ANG_OS_ERROR	270	troubles with operation system
ANG_DATA_ERROR	271	overflow at parameter values
ANG_PEAK_CNT_UFL	272	too less peaks
ANG_TIME_OUT	273	reading timeout
ANG_TOO_MANY_EXPOS	274	too many exposures wanted
ANG_PIX_CTRL_ERR	275	picture height out of range
ANG_MAX_POS_SKIP	276	positive exposure dynamic overflow
ANG_MAX_NEG_SKIP	277	negative exposure dynamic overflow
ANG_EXP_LIMIT	278	exposure time overflow
ANG_UNDER_EXPOSURE	279	picture underexposured
ANG_OVER_EXPOSURE	280	picture overexposured
ANG_TMANY_PEAKS	300	too many peaks detected
ANG_TLESS_PEAKS	301	too less peaks detected
ANG_PEAK_TOO_SLIM	302	peak too slim
ANG_PEAK_TOO_WIDE	303	peak to wide
ANG_BAD_PEAKDIFF	304	bad peak difference
ANG_UNDER_EXP_PICT	305	too less peak amplitude
ANG_PEAKS_INHOMOGEN	306	inhomogen peak amplitudes
ANG_NO_DECOD_POSS	307	no peak decoding possible

RetCodeName	Val	Description
ANG_UNSTABLE_DECOD	308	peak decoding not stable
ANG_TLESS_FPEAKS	309	too less valid finepeaks

A-3 ATA SUBSYSTEM

RetCodeName	Val	Description
ATA_RC_NOT_READY	512	ATR-System is not ready.
ATA_RC_NO_RESULT	513	Result isn't available yet.
ATA_RC_SEVERAL_TARGETS	514	Several Targets detected.
ATA_RC_BIG_SPOT	515	Spot is too big for analyse.
ATA_RC_BACKGROUND	516	Background is too bright.
ATA_RC_NO_TARGETS	517	No targets detected.
ATA_RC_NOT_ACCURAT	518	Accuracy worse than asked for.
ATA_RC_SPOT_ON_EDGE	519	Spot is on the edge of the sensing area.
ATA_RC_BLOOMING	522	Blooming or spot on edge detected.
ATA_RC_NOT_BUSY	523	ATR isn't in a continuous mode.
ATA_RC_STRANGE_LIGHT	524	Not the spot of the own target illuminator.
ATA_RC_V24_FAIL	525	Communication error to sensor (ATR).
ATA_RC_HZ_FAIL	527	No Spot detected in Hz-direction.
ATA_RC_V_FAIL	528	No Spot detected in V-direction.
ATA_RC_HZ_STRANGE_L	529	Strange light in Hz-direction.
ATA_RC_V_STRANGE_L	530	Strange light in V-direction.

A-4 EDM SUBSYSTEM

RetCodeName	Val	Description
EDM_COMERR	769	Communication with EDM failed
EDM_NOSIGNAL	770	no signal
EDM_PPM_MM	771	PPM and/or MM not zero
EDM_METER_FEET	772	EDM unit not set to meter
EDM_ERR12	773	battery low

RetCodeName	Val	Description
EDM_DIL99	774	limit at 99 measurements (DIL)

A-5 TMC SUBSYSTEM

RetCodeName	Val	Description
TMC_NO_FULL_CORRECTION	1283	Warning: measurement without full correction
TMC_ACCURACY_GUARANTEE	1284	Info: accuracy can not be guaranteed
TMC_ANGLE_OK	1285	Warning: only angle measurement valid
TMC_ANGLE_NO_FULL_CORRECTION	1288	Warning: only angle measurement valid but without full correction
TMC_ANGLE_ACCURACY_GUARANTEE	1289	Info: only angle measurement valid but accuracy can not be guarantee
TMC_ANGLE_ERROR	1290	Error: no angle measurement
TMC_DIST_PPM	1291	Error: wrong setting of PPM or MM on EDM
TMC_DIST_ERROR	1292	Error: distance measurement not done (no aim, etc.)
TMC_BUSY	1293	Error: system is busy (no measurement done)
TMC_SIGNAL_ERROR	1294	Error: no signal on EDM (only in signal mode)

A-6 MOT SUBSYSTEM

RetCodeName	Val	Description
MOT_RC_UNREADY	1792	Motorization not ready
MOT_RC_BUSY	1793	Motorization is handling another task
MOT_RC_NOT_OCONST	1794	Not in velocity mode
MOT_RC_NOT_CONFIG	1795	Motorization is in the wrong mode or busy
MOT_RC_NOT_POSIT	1796	Not in posit mode
MOT_RC_NOT_SERVICE	1797	Not in service mode
MOT_RC_NOT_BUSY	1798	Motorization is handling no task

RetCodeName	Val	Description
MOT_RC_NOT_LOCK	1799	Not in tracking mode

A-7 COM SUBSYSTEM

RetCodeName	Val	Description
RC_COM_ERO	3072	Initiate Extended Runtime Operation (ERO).
RC_COM_CANT_ENCODE	3073	Cannot encode arguments in client.
RC_COM_CANT_DECODE	3074	Cannot decode results in client.
RC_COM_CANT_SEND	3075	Hardware error while sending.
RC_COM_CANT_RECV	3076	Hardware error while receiving.
RC_COM_TIMEDOUT	3077	Request timed out.
RC_COM_WRONG_FORMAT	3078	Packet format error.
RC_COM_VER_MISMATCH	3079	Version mismatch between client and server.
RC_COM_CANT_DECODE_REQ	3080	Cannot decode arguments in server.
RC_COM_PROC_UNAVAIL	3081	Unknown RPC, procedure ID invalid.
RC_COM_CANT_ENCODE_REP	3082	Cannot encode results in server.
RC_COM_SYSTEM_ERR	3083	Unspecified generic system error.
RC_COM_FAILED	3085	Unspecified error.
RC_COM_NO_BINARY	3086	Binary protocol not available.
RC_COM_INTR	3087	Call interrupted.
RC_COM_REQUIRES_8DBITS	3090	Protocol needs 8bit encoded characters.
RC_COM_TR_ID_MISMATCH	3093	Transaction ID mismatch error.
RC_COM_NOT_GEOCOM	3094	Protocol not recognisable.
RC_COM_UNKNOWN_PORT	3095	(WIN) Invalid port address.

RetCodeName	Val	Description
RC_COM_ERO_END	3099	ERO is terminating.
RC_COM_OVERRUN	3100	Internal error: data buffer overflow.
RC_COM_SRVR_RX_CHECKSUM_ERROR	3101	Invalid checksum on server side received.
RC_COM_CLNT_RX_CHECKSUM_ERROR	3102	Invalid checksum on client side received.
RC_COM_PORT_NOT_AVAILABLE	3103	(WIN) Port not available.
RC_COM_PORT_NOT_OPEN	3104	(WIN) Port not opened.
RC_COM_NO_PARTNER	3105	(WIN) Unable to find TPS.
RC_COM_ERO_NOT_STARTED	3106	Extended Runtime Operation could not be started.
RC_COM_CONS_REQ	3107	Attention to send cons requests
RC_COM_SRVR_IS_SLEEPING	3108	TPS has gone to sleep. Wait and try again.
RC_COM_SRVR_IS_OFF	3109	TPS has shut down. Wait and try again.

A-8 WIR SUBSYSTEM

RetCodeName	Val	Description
WIR_PTNR_OVERFLOW	5121	point number overflow
WIR_NUM_ASCII_CARRY	5122	carry from number to ASCII conversion
WIR_PTNR_NO_INC	5123	can't increment point number
WIR_STEP_SIZE	5124	wrong step size
WIR_BUSY	5125	resource occupied
WIR_CONFIG_FNC	5127	user function selected
WIR_CANT_OPEN_FILE	5128	can't open file
WIR_FILE_WRITE_ERROR	5129	can't write into file
WIR_MEDIUM_NOMEM	5130	no anymore memory on PC-Card
WIR_NO_MEDIUM	5131	no PC-Card
WIR_EMPTY_FILE	5132	empty GSI file

RetCodeName	Val	Description
WIR_INVALID_DATA	5133	invalid data in GSI file
WIR_F2_BUTTON	5134	F2 button pressed
WIR_F3_BUTTON	5135	F3 button pressed
WIR_F4_BUTTON	5136	F4 button pressed
WIR_SHF2_BUTTON	5137	SHIFT F2 button pressed

A-9 AUT SUBSYSTEM

RetCodeName	Val	Description
AUT_RC_TIMEOUT	8704	Position not reached
AUT_RC_DETENT_ERROR	8705	Positioning not possible due to mounted EDM
AUT_RC_ANGLE_ERROR	8706	Angle measurement error
AUT_RC_MOTOR_ERROR	8707	Motorization error
AUT_RC_INCACC	8708	Position not exactly reached
AUT_RC_DEV_ERROR	8709	Deviation measurement error
AUT_RC_NO_TARGET	8710	No target detected
AUT_RC_MULTIPLE_TARGETS	8711	Multiple target detected
AUT_RC_BAD_ENVIRONMENT	8712	Bad environment conditions
AUT_RC_DETECTOR_ERROR	8713	Error in target acquisition
AUT_RC_NOT_ENABLED	8714	Target acquisition not enabled
AUT_RC_CALACC	8715	ATR-Calibration failed
AUT_RC_ACCURACY	8716	Target position not exactly reached

A-10 BAP SUBSYSTEM

RetCodeName	Val	Description
BAP_CHANGE_ALL_TO_DIST	9217	Command changed from ALL to DIST

B HARDWARE INTERFACE

B-1 SERIAL INTERFACE

A RS-232 interface is used as a hardware link between the TPS1000 and an external computer.

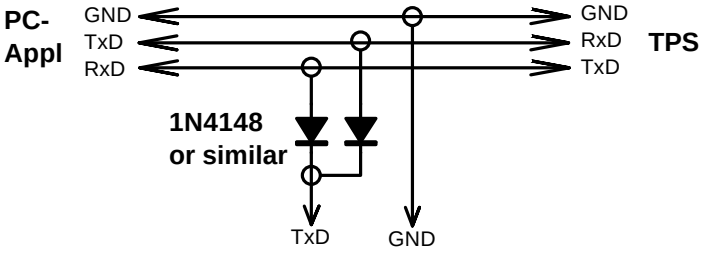
Signal paths	RxD	
	TxD	
	Signal Ground	
Voltage levels	Logical 0 +3V to +25V	
	Logical 1 -3V to -25V	
Baud rate	2400	
	4800	
	9600	Default
	19200	
Parity	None	Fixed
Data bits	8	Fixed
Stop bits	1	Fixed
Terminator	CR/LF	Default

The default settings for the interface are 9600 Baud, 8 data bits, 1 stop bit, no parity. The communication terminator is set to CR/LF. The parameters marked as 'Fixed' may not be changed. The other parameters are variable may be changed by the user.

B-2 DEBUGGING UTILITY

When debugging communicating systems it may be hard to locate the source of an error. Especially in combination with radios to communicate wireless, the number of error sources increases. The following should be checked carefully therefore:

- Are all communication parameters set up properly? Do both participants share the same parameters?
- Have the serial buffer been flushed after opening the serial port? If not and you are using the ASCII protocol then use a leading <LF> to clear the receiver buffer. In the function call protocol you do not need to take care of that.
- When using the ASCII protocol: Is your implementation of the protocol flow indeed synchronous? Or are you sending requests before having received the last reply?
- Are handshake lines for the radios set correctly?
- In case of character errors check shielding of the radio wiring and potential buffer overflow. In case of Windows on 386 and 486 computers, check the UART type. If you do not have a UART with built in buffers (16550 type), you may loose characters too.



It may be helpful for debugging purposes to build up a special cable to monitor the data transfers.

C PROVIDED SAMPLES

C-1 SETTINGS FOR TERMINAL EMULATOR

To see how ASCII protocol works take a closer look at the provided settings file for the application Terminal.exe.

Use the terminal emulator Terminal.exe which has been provided by Microsoft with MS-Windows 3.1/3.11 and use `geocom.trm` as the set up file for it.

Connect the TPS1000 instrument to the PC by using a standard GSI connection cable. Start the emulator. `Geocom.trm` has been set up to the default parameters of GeoCOM (9600 Baud, 8 bit, no parity, 1 stop bit). Be sure that the instrument and the terminal emulator use the same settings and switch the instrument to online mode. If not already enabled then enable the function key bar of the application.

Then you can use the emulator to send requests by pressing the assigned buttons. The replies will appear in the emulator's window.

C-2 PROGRAM FRAMES

C-2.1 VBA Sample Program

The sample program shows how simple it is to build an effective application with Visual Basic. The sample program represents a simple measurement task that measures and displays the Hz angle and the V angle continuously. In addition you have the possibility to perform a distance measurement with the following distance measurement programs: single distance standard, single distance fast and tracking. In order to execute this example program, install MSVB4.0 or MSVB5.0 on your hard disk and copy the following files in a directory of your choice:

`\SAMPLES\VB\VBSAMPLE.VBP`	Visual Basic Project of the sample.
`\SAMPLES\VB\VBSAMPLE.FRM`	Frame of the sample, contains the real basic code (functions, declarations,...)
`\SAMPLES\VB\VBSAMP16.EXE`	Executable 16 bit Visual Basic sample program
`\SAMPLES\VB\VBSAMP32.EXE`	Executable 32 bit Visual Basic sample program

\DEV\VB\STUPS_P.BAS	Contains the declarations of the TPS1000 system functions for the 16 bit version.
\DEV\VB\STUPS32P.BAS	32 bit version of STUPS_P.BAS.
\DEV\WIN16\GEOCOM.DLL	Contains the implementation of 16 bit GeoCOM version
\DEV\WIN32\GEOCOM32.DLL	32 bit version of GEOCOM.DLL

Finally connect the TPS1000 Theodolite with the preferred serial port on your personal computer and invoke the executable file. Select the corresponding port and start the application with the button START. The button QUIT terminates the application.

Note: The application works only with 19200 baud.

C-2.2 C/C++ Sample Programs

The provided sample programs show simple Visual C++ MFC (Microsoft foundation classes) applications. The same samples are available as 16bit- and 32bit-MDI (Multi Document-Interface) applications. The functionality is exactly the same as in the Visual Basic program above.

The following files have to be copied into a Visual C++ Version 1.5 (or later) working directory in order to build a 16bit application:

\SAMPLES\C16\CSAMP16.DEF	Definition file of the 16bit project
\SAMPLES\C16\CSAMP16.MAK	Make file of the 16bit project
\SAMPLES\C16*.CPP	C++ source files
\SAMPLES\C16*.H	C++ header files
\SAMPLES\C16\CSAMP16.RC	Resource file
\SAMPLES\C16\RES*.*	Resource files
\SAMPLES\C16\CTL3D.LIB and CTL3D.DLL	Windows system files
\SAMPLES\C16\CSAMP16.EXE	Executable 16bit sample program
\DEV\WIN16\GEOCOM.DLL	Implementation of 16 bit GeoCOM version
\DEV\WIN16\GEOCOM.LIB	Library of 16 bit GeoCOM version

\DEV\WIN16\COM_PUB.HPP	Header file for GeoCOM

The following files have to be copied into a Visual C++ Version 4.0 (or later) working directory in order to build a 32bit application:

\SAMPLES\C32\CSAMP32.DSW	Work space file of the 32bit project
\SAMPLES\C32*.CPP	C++ source files
\SAMPLES\C32*.H	C++ header files
\SAMPLES\C32\CSAMP16.RC	Resource file
\SAMPLES\C32\RES*.*	Resource files
\SAMPLES\C32\CSAMP32.EXE	Executable 32bit sample program
\DEV\WIN32\GEOCOM32.DLL	32 bit version of GEOCOM.DLL
\DEV\WIN32\GEOCOM32.LIB	Library of 32 bit GeoCOM version
\DEV\WIN32\COM_PUB.HPP	Header file for GeoCOM

Note: Most of the 32bit-project-files are identical to the 16bit-project-files.

D LIST OF RPC'S

D-1 ALPHA ORDER

Name	Number	Page
AUT_ChangeFace4	9028	6-16
AUT_FineAdjust3	9037	6-19
AUT_GetATRStatus	9019	6-2
AUT_GetFineAdjustMode	9030	6-25
AUT_GetLockStatus	9021	6-4
AUT_LockIn	9013	6-27
AUT_MakePositioning4	9027	6-11
AUT_ReadTimeout	9012	6-9
AUT_ReadTol	9008	6-7
AUT_Search2	9029	6-22
AUT_SetATRStatus	9018	6-4
AUT_SetFineAdjustMode	9031	6-26
AUT_SetLockStatus	9020	6-6
AUT_SetTimeout	9011	6-11
AUT_SetTol	9007	6-8
BAP_GetLastDisplayedError	17003	7-1
BAP_MeasDistanceAngle	17017	7-2
BMM_BeepAlarm	11004	8-1
BMM_BeepNormal	11003	8-2
BMM_BeepOff	11002	8-3
BMM_BeepOn	11001	8-2
COM_CloseConnection		5-8
COM_EnableSignOff	115	9-6
COM_End		5-5
COM_GetBaudRate		5-9
COM_GetBinaryAvailable	113	9-7
COM_GetComFormat		5-13
COM_GetDoublePrecision	108	5-2
COM_GetErrorText		5-18
COM_GetSWVersion	110	9-1
COM_GetTimeOut		5-11
COM_GetTPSState		5-20
COM_GetWinSWVersion		5-19
COM_Init		5-5
COM_Local	1	9-3
COM_NullProc	0	9-5

Name	Number	Page
COM_OpenConnection		5-6
COM_SetBaudRate		5-10
COM_SetBinaryAvailable	114	9-8
COM_SetComFormat		5-14
COM_SetConnDlgFlag		5-16
COM_SetDoublePrecision	107	5-4
COM_SetSendDelay	109	9-2
COM_SetTimeOut		5-12
COM_SwitchOffTPS	112	9-5
COM_SwitchOnTPS	111	9-4
COM_UseWindow		5-15
COM_ViewError		5-17
CSV_GetDateTime	5008	10-7
CSV_GetDeviceConfig	5035	10-6
CSV_GetInstrumentName	5004	10-4
CSV_GetInstrumentNo	5003	10-3
CSV_GetIntTemp	5011	10-12
CSV_GetSWVersion2	5034	10-9
CSV_GetUserInstrumentName	5006	10-4
CSV_GetVBat	5009	10-10
CSV_GetVMem	5010	10-11
CSV_SetDateTime	5007	10-8
CSV_SetUserInstrumentName	5005	10-5
CTL_GetUpCounter	12003	11-1
EDM_GetBumerang	1044	12-5
EDM_GetTrkLightBrightness	1041	12-8
EDM_GetTrkLightSwitch	1040	12-10
EDM_Laserpointer	1004	12-2
EDM_On	1010	12-4
EDM_SetBumerang	1007	12-7
EDM_SetTrkLightBrightness	1032	12-9
EDM_SetTrkLightSwitch	1031	12-10
MOT_ReadLockStatus	6021	13-2
MOT_SetVelocity	6004	13-5
MOT_StartController	6001	13-3
MOT_StopController	6002	13-4
SUP_GetConfig	14001	14-2
SUP_SetConfig	14002	14-3
SUP_SwitchLowTempControl	14003	14-4
TMC_DoMeasure	2008	15-21
TMC_GetAngle1	2003	15-12
TMC_GetAngle5	2107	15-14

Name	Number	Page
TMC_GetAngSwitch	2014	15-40
TMC_GetAtmCorr	2029	15-27
TMC_GetCoordinate1	2082	15-6
TMC_GetEdmMode	2021	15-43
TMC_GetFace	2026	15-38
TMC_GetHeight	2011	15-25
TMC_GetInclineSwitch	2007	15-41
TMC_GetPrismCorr	2023	15-30
TMC_GetRefractiveCorr	2031	15-32
TMC_GetRefractiveMethod	2091	15-34
TMC_GetSignal	2022	15-39
TMC_GetSimpleCoord	2116	15-45
TMC_GetSimpleMea	2108	15-9
TMC_GetStation	2009	15-36
TMC_IfDataAzeCorrError	2114	15-49
TMC_IfDataIncCorrError	2115	15-50
TMC_QuickDist	2117	15-16
TMC_SetAngSwitch	2016	15-51
TMC_SetAtmCorr	2028	15-27
TMC_SetEdmMode	2020	15-44
TMC_SetHandDist	2019	15-23
TMC_SetHeight	2012	15-26
TMC_SetInclineSwitch	2006	15-42
TMC_SetOrientation	2113	15-28
TMC_SetPrismCorr	2024	15-31
TMC_SetRefractiveCorr	2030	15-33
TMC_SetRefractiveMethod	2090	15-35
TMC_SetStation	2010	15-37
WIR_GetRecFormat	8011	16-1
WIR_SetRecFormat	8012	16-2

D-2 NUMERIC ORDER

Name	Number	Page
COM_CloseConnection		5-8
COM_End		5-5
COM_GetBaudRate		5-9
COM_GetComFormat		5-13
COM_GetErrorText		5-18
COM_GetTimeOut		5-11

Name	Number	Page
COM_GetTPSState		5-20
COM_GetWinSWVersion		5-19
COM_Init		5-5
COM_OpenConnection		5-6
COM_SetBaudRate		5-10
COM_SetComFormat		5-14
COM_SetConnDlgFlag		5-16
COM_SetTimeOut		5-12
COM_UseWindow		5-15
COM_ViewError		5-17
COM_NullProc	0	9-5
COM_Local	1	9-3
COM_SetDoublePrecision	107	5-4
COM_GetDoublePrecision	108	5-2
COM_SetSendDelay	109	9-2
COM_GetSWVersion	110	9-1
COM_SwitchOnTPS	111	9-4
COM_SwitchOffTPS	112	9-5
COM_GetBinaryAvailable	113	9-7
COM_SetBinaryAvailable	114	9-8
COM_EnableSignOff	115	9-6
EDM_Laserpointer	1004	12-2
EDM_SetBumerang	1007	12-7
EDM_On	1010	12-4
EDM_SetTrkLightSwitch	1031	12-10
EDM_SetTrkLightBrightness	1032	12-9
EDM_GetTrkLightSwitch	1040	12-10
EDM_GetTrkLightBrightness	1041	12-8
EDM_GetBumerang	1044	12-5
TMC_GetAngle1	2003	15-12
TMC_SetInclineSwitch	2006	15-42
TMC_GetInclineSwitch	2007	15-41
TMC_DoMeasure	2008	15-21
TMC_GetStation	2009	15-36
TMC_SetStation	2010	15-37
TMC_GetHeight	2011	15-25
TMC_SetHeight	2012	15-26
TMC_GetAngSwitch	2014	15-40
TMC_SetAngSwitch	2016	15-51
TMC_SetHandDist	2019	15-23
TMC_SetEdmMode	2020	15-44
TMC_GetEdmMode	2021	15-43

Name	Number	Page
TMC_GetSignal	2022	15-39
TMC_GetPrismCorr	2023	15-30
TMC_SetPrismCorr	2024	15-31
TMC_GetFace	2026	15-38
TMC_SetAtmCorr	2028	15-27
TMC_GetAtmCorr	2029	15-27
TMC_SetRefractiveCorr	2030	15-33
TMC_GetRefractiveCorr	2031	15-32
TMC_GetCoordinate1	2082	15-6
TMC_SetRefractiveMethod	2090	15-35
TMC_GetRefractiveMethod	2091	15-34
TMC_GetAngle5	2107	15-14
TMC_GetSimpleMea	2108	15-9
TMC_SetOrientation	2113	15-28
TMC_IfDataAzeCorrError	2114	15-49
TMC_IfDataIncCorrError	2115	15-50
TMC_GetSimpleCoord	2116	15-45
TMC_QuickDist	2117	15-16
CSV_GetInstrumentNo	5003	10-3
CSV_GetInstrumentName	5004	10-4
CSV_SetUserInstrumentName	5005	10-5
CSV_GetUserInstrumentName	5006	10-4
CSV_SetDateTime	5007	10-8
CSV_GetDateTime	5008	10-7
CSV_GetVBat	5009	10-10
CSV_GetVMem	5010	10-11
CSV_GetIntTemp	5011	10-12
CSV_GetSWVersion2	5034	10-9
CSV_GetDeviceConfig	5035	10-6
MOT_StartController	6001	13-3
MOT_StopController	6002	13-4
MOT_SetVelocity	6004	13-5
MOT_ReadLockStatus	6021	13-2
WIR_GetRecFormat	8011	16-1
WIR_SetRecFormat	8012	16-2
AUT_SetTol	9007	6-8
AUT_ReadTol	9008	6-7
AUT_SetTimeout	9011	6-11
AUT_ReadTimeout	9012	6-9
AUT_LockIn	9013	6-27
AUT_SetATRStatus	9018	6-4
AUT_GetATRStatus	9019	6-2

Name	Number	Page
AUT_SetLockStatus	9020	6-6
AUT_GetLockStatus	9021	6-4
AUT_MakePositioning4	9027	6-11
AUT_ChangeFace4	9028	6-16
AUT_Search2	9029	6-22
AUT_GetFineAdjustMode	9030	6-25
AUT_SetFineAdjustMode	9031	6-26
AUT_FineAdjust3	9037	6-19
BMM_BeepOn	11001	8-2
BMM_BeepOff	11002	8-3
BMM_BeepNormal	11003	8-2
BMM_BeepAlarm	11004	8-1
CTL_GetUpCounter	12003	11-1
SUP_GetConfig	14001	14-2
SUP_SetConfig	14002	14-3
SUP_SwitchLowTempControl	14003	14-4
BAP_GetLastDisplayedError	17003	7-1
BAP_MeasDistanceAngle	17017	7-2